Secure Provisioning of Cloud Services
based on SLA Management

SPECS Project - Deliverable 4.3.3

Implementation of the enforcement SLA components - Finalized

Version no. 1.1
19 July 2016

The activities reported in this deliverable are partially supported
by the European Community's Seventh Framework Programme under grant agreement no. 610795.

Deliverable information

Deliverable no.:	D4.3.3
Deliverable title:	Implementation of the enforcement SLA components – Finalized
Deliverable nature:	Prototype
Dissemination level:	Public
Contractual delivery:	19 July 2016
Actual delivery date:	19 July 2016
Author(s):	Jolanda Modic (XLAB), Miha Stopar (XLAB)
Contributors:	Alessandra de Benedictis (CeRICT), Massimiliano Rak (CeRICT)
Reviewers:	Dana Petcu (IeAT), Stefano Marrone (CeRICT)
Task contributing to the deliverable:	T4.3
Total number of pages:	70

Executive summary

The main focus of this deliverable is a presentation of the final implementation of the Enforcement module. The demonstrated prototypes are based on the following:

- The requirements and design reported in D4.1.2 and D4.2.2, respectively.
- Prototypes presented at previous milestones and described in D4.3.1 and D4.3.2.
- The final SPECS flow and SPECS integration activities discussed in D1.1.3 and deliverables of the task T1.5 (D1.5.1 and D1.5.2), respectively.

In particular, this document presents:

- Demonstration of the main Enforcement components: We report improvements with respect to M24 (D4.3.2) and present implementation details related to the main Enforcement components that have been finalized at M30.
- Demonstration of the SPECS security mechanisms: We demonstrate prototypes of security mechanisms that have been improved with respect to M24 (D4.3.2) and finalized at M30.
- Analysis of the developed software: We report results of the performance, scalability, and security reviews of the designed and developed software.

Table of contents

Index of figures

Index of tables

1. Introduction

In SPECS, the SLA life cycle is defined as follows (and depicted in Figure 1). The End-user accesses the SPECS application and starts the negotiation process. During this phase, the End-user defines the required cloud service and the security features that she/he wishes to have enforced and monitored on top of it. After the negotiation process is completed (and an SLA is signed), the implementation phase starts, which comprises (i) the step of analysing the signed SLA to determine the number of cloud resources to acquire, (ii) the step of extracting the security features required by End-user to determine security mechanisms (and their configurations) to deploy on top of the acquired resources, and (iii) the actual acquisition and deployment. After the SLA is implemented the SLA monitoring phase starts. If at any given point the Monitoring module detects a suspicious behaviour, the Enforcement module is notified and the remediation process is activated. During this phase, each potential or actual SLA violation has to be analysed, the root cause has to be identified, and actions to be taken to prevent or recover from the violation have to be determined. The remediation process can then either end with a success (the potential violation is mitigated and the SLA re-enters the monitoring phase), with a reconfiguration (the actual violation can be recovered from with a reconfiguration of the cloud service) or renegotiation (we are unable to automatically handle the violation).

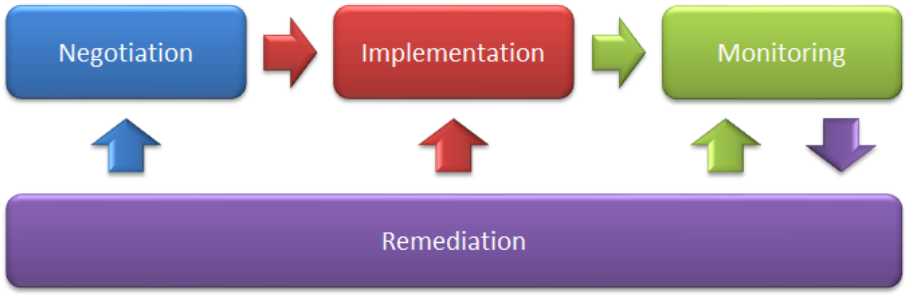


Figure 1. SLA phases

The Enforcement module in SPECS orchestrates two main steps of the SLA life-cycle, namely SLA implementation and SLA remediation. During the SLA negotiation phase, the Enforcement module also supports the generation of valid supply chains according to End-user's security requirements, Cloud Service Providers' (CSPs') capabilities, and possible SPECS security enhancements.

To this end, the Enforcement module comprises a set of main components (supporting the SLA life cycle) and a set of security mechanisms which can enhance security level of the acquired cloud service. In this document, we present the final prototypes of the Enforcement module. With respect to previous iterations, we report changes and improvements, and present mechanisms not yet developed at previous milestones. Note that all testing aspects are reported in deliverable D4.5.3.

The document is structured as follows. In Section 2 we report about deliverables of other tasks that either served as an input or took this document as the input for its activities. Then we focus on presenting prototypes of main Enforcement components (Section 3) and SPECS security mechanisms (Section 4). The document is concluded with a short summary of results compared with the current state-of-the-art in Section 5.

2. Relationship with other deliverables

The final Enforcement prototypes presented in this deliverable comprise a set of core components, which orchestrate the enforcement process (SLA implementation, SLA remediation), and a set of security mechanisms that enforce and monitor negotiated SLAs. Due to the highly modular architecture of the SPECS framework, the input for the task T4.3 came from almost all workpackages:

- **WP1:** Final architecture of the SPECS framework (D1.1.3), designed interfaces (D1.3), and integration (D1.5.1).
- **WP2:** Final implementation of the negotiation process (D2.3.2, D2.3.3).
- **WP3:** Final implementation of the monitoring process (D3.4.2) and the OpenVAS prototype (D3.4.1).
- **WP4:** The design (D4.2.2) and intermediate prototype of the Enforcement module (D4.3.2).
- **WP5:** Validation scenarios (D5.1.2) and Secure Storage application, which integrates two (in task T5.2 improved) security mechanisms (D5.2.1, D5.2.2).

The results of the task T4.3 also served as an input to other workpackages. Namely:

- **WP1:** Integration activities (D1.5.1, D1.5.2).
- **WP2:** Final implementation of the negotiation module (D2.3.2, D2.3.3).
- **WP3:** Final implementation of the monitoring module (D3.4.2).
- **WP4:** Testing of the Enforcement module (D4.5.3).
- **WP5:** The Secure Storage application (D5.2.1, D5.2.2), the ngDC application (D5.3), the AAAaaS application (D5.4), which integrate components and security mechanisms presented in this deliverable.
- **WP6:** The defined security metrics as an input for standardisation activities (D6.2.3).

All mentioned relationships are detailed in Figure 2.

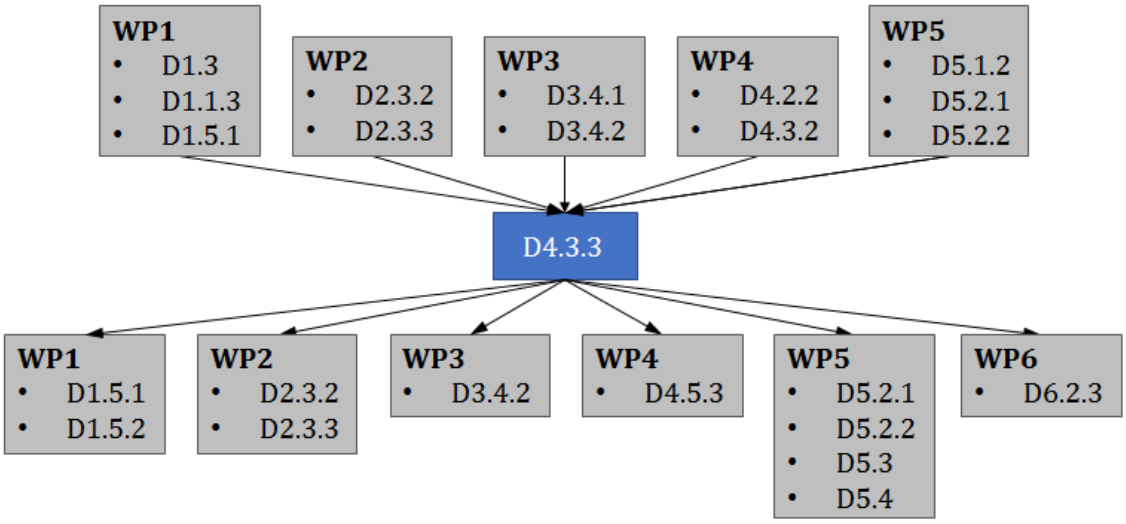


Figure 2. Relationship with other deliverables

3. Main Enforcement components

The Enforcement module in SPECS is a system that orchestrates two phases of the SLA life cycle, namely the SLA implementation and the SLA remediation phase. Moreover, the Enforcement module supports the SLA negotiation phase by verifying the feasibility of SLA offers. To this end, the Enforcement module comprises the following main components:

- **Planning**: Supports the SLA negotiation phase by generating supply chains for End-user's security requirements, and builds implementation and reaction plans for signed and renegotiated/terminated SLAs, respectively.
- **Implementation** integrated with the **Broker** and the **Chef Server**: Acquires resources and deploys and configures (or reconfigures or terminates) security mechanisms and configures (or reconfigures) monitoring components according to the implementation or reaction plan. The Broker manages acquisition of resources and the Chef Server oversees automated management of their configurations.
- **Diagnosis**: Classifies and analyses detected monitoring events (potential and actual SLA violations).
- **Remediation Decision System component (RDS)**: Prepares remediation plans according to results of the diagnosis process.

The high level architecture of the Enforcement module is presented in Figure 3.

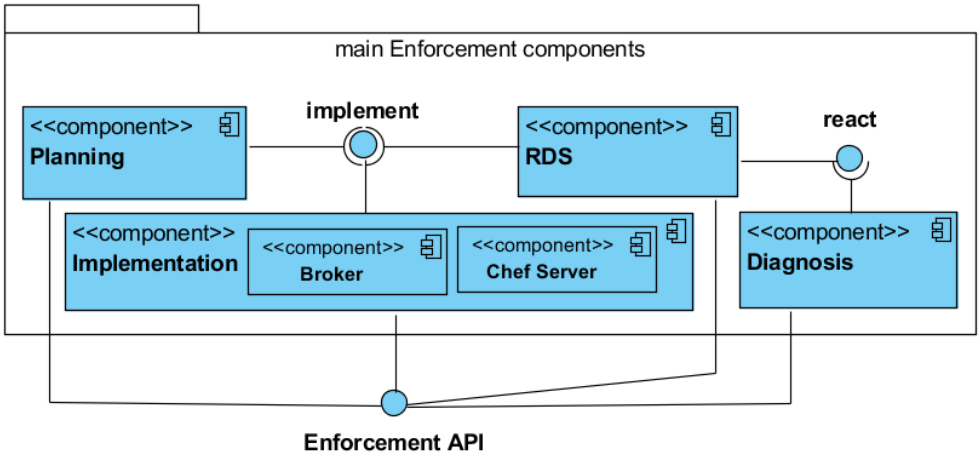


Figure 3. Architecture of the Enforcement module (main Enforcement components)

Before we present the final status of implementation activities and report changes with respect to deliverable D4.3.2, we briefly summarize the processes orchestrated by the main Enforcement components presented in deliverable D4.3.2.

3.1. SLA implementation phase

During the SLA negotiation phase, the End-user expresses her/his security requirements. The Enforcement module is responsible for (i) verifying that the set of security requirements is feasible in order to build valid SLA Offers and (ii) acquiring and configuring cloud resources according to the signed SLA.

To this end, the Enforcement module comprises two components which implement a set of functionalities enabling the SLA implementation phase (see Figure 4). When the Negotiation module gathers all security requirements from the End-user, it invokes the Planning

component to generate valid supply chains which form a base for creating SLA Offers (*Step 1*). If the End-user later selects one of the offered SLAs and signs it, the Negotiation module triggers the Enforcement module to implement it (*Step 2*). The Planning component prepares the implementation plan and passes it to the Implementation component (*Step 3*), and configures the Monitoring module (*Step 4*). Finally, the Implementation component acquires resources specified in the implementation plan and deploys and configures all mechanisms needed to enforce and monitor the security features selected by the End-user and specified in the SLA (*Step 5*).

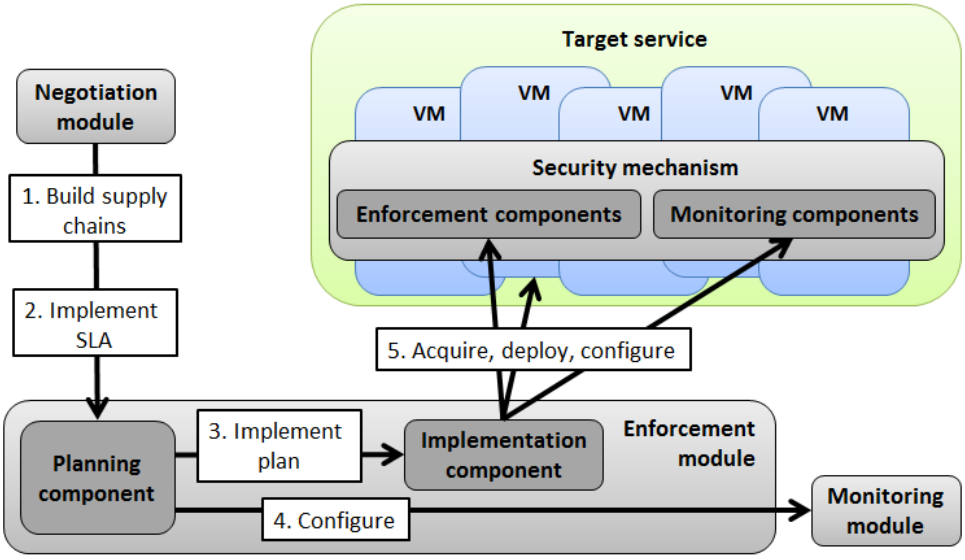


Figure 4. SLA implementation phase

Generation of supply chains and preparation and execution of implementation plans is further elaborated in the next subsections.

3.1.1. Generation of supply chains

A supply chain in SPECS is a combination of cloud resources (CSP, zone, instance type, etc.) and components of security mechanisms needed to enforce and monitor an SLA. According to security preferences of the End-user specified in her/his SLA, the Planning component has to determine (i) the set of mechanisms required for implementation of an SLA, (ii) the set of components of each of the required mechanisms, and (iii) the number of instances of each required component. Moreover, the Planning component has to determine the number of resources on which the components of security mechanisms have to be deployed and determine the distribution of the required components over all acquired resources.

To solve this so-called planning problem and build valid supply chains, the Planning component uses input from three sources:

- **End-user**: The SLA specifies what needs to be enforced and monitored.
- **SPECS**: The mechanisms' metadata specifies which features are enforced and/or monitored by which mechanism and how (with which components and how these components need to be deployed).
- **CSP**: Resource information for each CSP specifies zones, VM types, maximum number of acquirable VMs per zone, etc.

The Planning component thus has to deal with three different sets of constraints to determine the right mechanisms to implement an SLA and to find an optimal allocation for them. To this end, the Planning component uses an innovative algorithm developed in this project [6], to model and solve the planning problem. The solution is a set of supply chains, each specifying the resource information (CSP, zone, VM type) and allocation of components (number of VMs to acquire, allocation of components over them).

The process of generating supply chains is depicted in Figure 5 and presented with an example below.

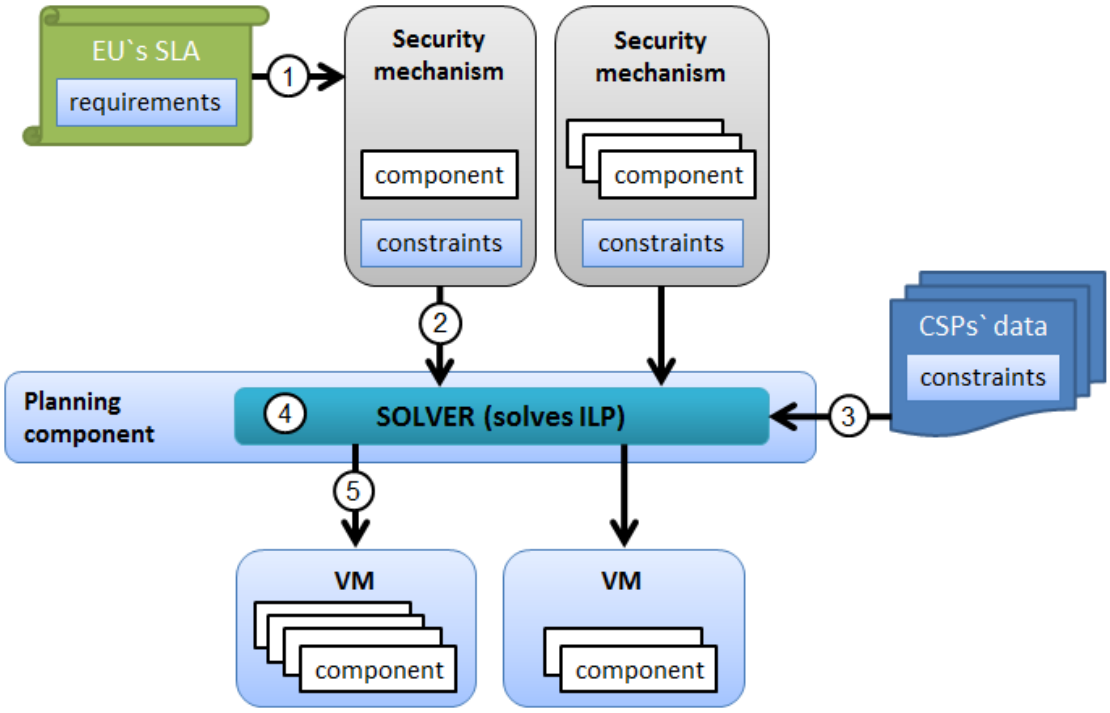


Figure 5. Building supply chains

Let us assume that an End-user aims at acquiring a secure web container. The End-user requires a redundant web server (*Level of Redundancy (LoR) = 2*) implemented with two different softwares (*Level of Diversity (LoD) = 2*). Additionally, the End-user negotiates software vulnerability assessment in the form of periodic vulnerability scans (*Scanning Frequency (SF) = 24h*). The Planning component parses the SLA Template which specifies End-user's requirements (*Step 1*) and identifies two security mechanisms to enforce and monitor these features, namely the WebPool (for *LoR* and *LoD*) and the SVA (for *SF*). From the Service Manager component (element of the SLA Platform; see deliverable D1.4.1) it retrieves the metadata for these two mechanisms to get information about the components that have to be deployed and constraints associated to them (*Step 2*). For example, to cover requirements associated to redundancy and diversity, SPECS needs to deploy two different web servers (Apache and Nginx) and a load balancer (HAProxy) on three different VMs. To enable the vulnerability assessment, a Scanner needs to be deployed on each VM hosting a web server and the Dashboard to collect and present scanning reports needs to be deployed on the VM hosting the load balancer. According to these constraints and the constraints coming from the resource information specified in the SLA Template (*Step 3*), the Planning component models

and solves the planning problem (*Step 4*). Assuming that there are two CSPs with different VM types available, the Planning component builds the following two supply chains (*Step 5*):

- **Supply Chain 1**:
  - Resources: 3 VMs with Amazon (zone: eu-west-1, VM type: t2.medium)
  - Allocation: {HAProxy, Dashboard} + {Apache, Scanner} + {Nginx, Scanner};
- **Supply Chain 2**:
  - Resources: 4 VMs with Gcloud (zone: europe-west1-b, VM type: n1-standard-1)
  - Allocation: {HAProxy} + {Dashboard} + {Apache, Scanner} + {Nginx, Scanner}.

The generated supply chains are later transformed into a set of SLA Offers (for details see deliverable D2.3.3). If the End-user accepts and signs one of the SLA Offers, the Enforcement module is triggered again to implement it. The process of generating the implementation plan for a signed SLA is discussed in the next section.

3.1.2. Generation of implementation plans

When the negotiation process is complete and the End-user signs an SLA, the Enforcement module is triggered to implement it. To this end, the Planning component has to build an implementation plan which automates the acquisition of resources and deployment and configuration of security mechanisms.

The implementation plan specifies the following:

- **Resources**:
  - CSPs: The ID of the CSP and the zone in which the VMs should be acquired.
  - VMs: What VM type and how many of them should be acquired.
  - Allocation: Which SPECS components need to be deployed and on which VMs.
- **Configuration details**:
  - Enforcement actions: What and how to enforce (which metrics/SLOs should be enforced and how, for example, in order to enforce the required level of redundancy metric/SLO, enough web servers need to be deployed).
  - Monitoring actions: What and how to monitor (which metrics/SLOs should be monitored and how, for example, in order to monitor the level of redundancy, monitoring adapters have to continuously evaluate the measurements associated to responsiveness of deployed web servers). As discussed in deliverable D4.3.2, each metric in SPECS is associated to one or more measurements which are continuously evaluated to verify the fulfilment of the commitments in the SLA.
  - Frequencies: When to enforce and monitor (for example, how often to execute vulnerability scans or how often to check responsiveness of web servers).
  - Remediation actions: Which events are considered as potential/actual SLA violations and how to react in case of their detection (for example, unresponsiveness of a vulnerability scanner is considered as an SLA alert which we resolve by restarting/reinstalling the scanner, and the age of the scanning report, which is higher than the required frequency of vulnerability scans, is considered as an SLA violation which is resolved by immediately activating a new scan and possibly reinstalling the scanner).

The Planning component is also responsible for the configuration of the Monitoring module which analyses all measurements taken during runtime and filters out all potential and actual SLA violations (for the details about the monitoring process see deliverable D3.4.2). The so called SLA with alerts is simply the signed SLA enriched with SLOs that are defined according to the measurements associated to each existing metric/SLO in the SLA.

The detailed process of generating implementation plans and SLAs with alerts is depicted in Figure 6 and presented below with an example.

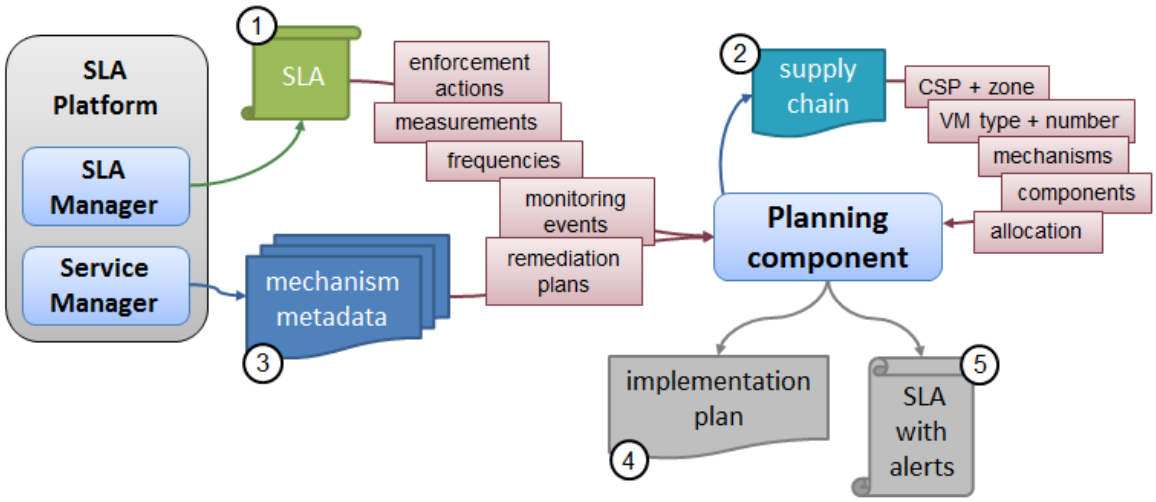


Figure 6. Building implementation plans and SLAs with alerts

Let us take a simple example where the End-user requires a web server with periodic vulnerability scans executed every 24h. When the End-user signs an SLA with these specifications, the Planning component retrieves the SLA, the associated supply chain, and the metadata information for all mechanisms that need to be deployed in order to implement the SLA. The SLA specifies one single SLO *Scanning Frequency (SF) = 24h*, which requires deployment of the SVA security mechanism. In particular, the generated supply chain assumes deployment of 2 VMs where one VM hosts the web server, the Scanner component which performs scans, the SVA Enforcement component which manages lists of published vulnerabilities, and the SVA Monitoring component which monitors all measurements associated to the SF metric, and the other VM hosts the SVA Dashboard which presents scanning results to the End-user. According to the metadata for the SVA mechanism, the Planning component identifies all enforcement, monitoring, and remediation actions defined for these components and the SF metric, and generates the implementation plan and the SLA with alerts.

The execution of the implementation plans, which is orchestrated by the Implementation component, is discussed in the next section, whereas the configuration of the Monitoring module with the SLA with alerts is presented in deliverable D3.4.2.

3.1.3. Execution of implementation plans

The SLA implementation in SPECS is automated with the Implementation component. More precisely, the SLA implementation is orchestrated by the Implementation component which

integrates the Broker component to acquire external cloud resources and the Chef Server which stores implementation plans and automates deployment and configurations.
As introduced in deliverable D4.2.2, all deployment and configuration actions are managed with Chef [7], which is the leading configuration and management tool. Chef uses so-called *recipes* to automate infrastructure tasks. The implementation plan specifies which components need to be deployed on acquired VMs (in terms of which recipes need to be run on each VM) and recipes specify how components need to be deployed and how they are to be configured and managed.

Recipes for all mechanisms used in SPECS are organized in *cookbooks* and available on the Bitbucket [5].

The process of executing implementation plans is defined as follows (see Figure 7). The Implementation component upon the invocation of the Planning component retrieves the implementation plan generated by the Planning component for an SLA from the Chef Server (*Step 1*). According to the resources specified in the implementation plan, the Implementation component triggers the Broker to acquire cloud resources (*Step 2*). Then the list of recipes that need to be run on the acquired resources is extracted from the implementation plan and the Chef Server is triggered to run them in order to deploy and configure security mechanisms (*Step 3*). When all services are set and running, the Implementation component updates the implementation plan (e.g., adding IPs of the acquired resources) and stores it in the Chef Server (*Step 4*). When this process is complete, the SLA is implemented and the monitoring phase starts.

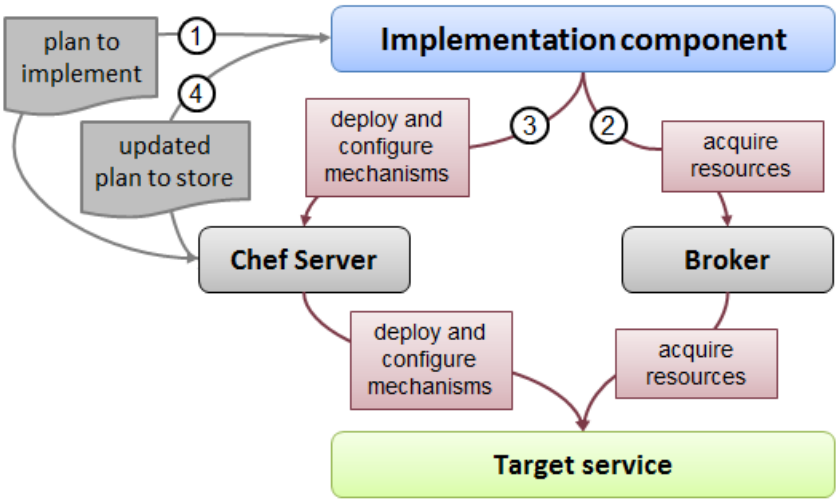


Figure 7. Executing implementation plans

If the Monitoring module at any given point detects a potential or an actual SLA violation, the remediation process starts. The details of the remediation phase are discussed in the next section.

3.2. SLA remediation phase

During the SLA remediation phase, the Enforcement module is responsible for (i) analysis of the notified monitoring events, (ii) identification of remediation actions to be applied in order to prevent or recover from an SLA violation, and (iii) execute the identified remediation plan.

To this end, the Diagnosis, the Remediation Decision System (RDS), and the Implementation components of the Enforcement module implement a set of functionalities which enable the SLA remediation phase (see Figure 8). The Monitoring module continuously collects the monitoring data from the adaptors deployed on cloud resources (*Step 1*). When an SLA alert or an SLA violation is detected, the Monitoring module immediately notifies the Enforcement module (*Step 2*). The Diagnosis component first analyses the notification to determine whether the event is an alert or a violation and to determine the causing factors (*Step 3*), and afterwards the RDS component identifies the set of remediation actions to apply in order to mitigate the risk of having an SLA violation or to recover from it (*Step 4*). Finally, the identified remediation plan, which mainly comprises reconfigurations of the target service, is executed by the Implementation component (*Step 5*).

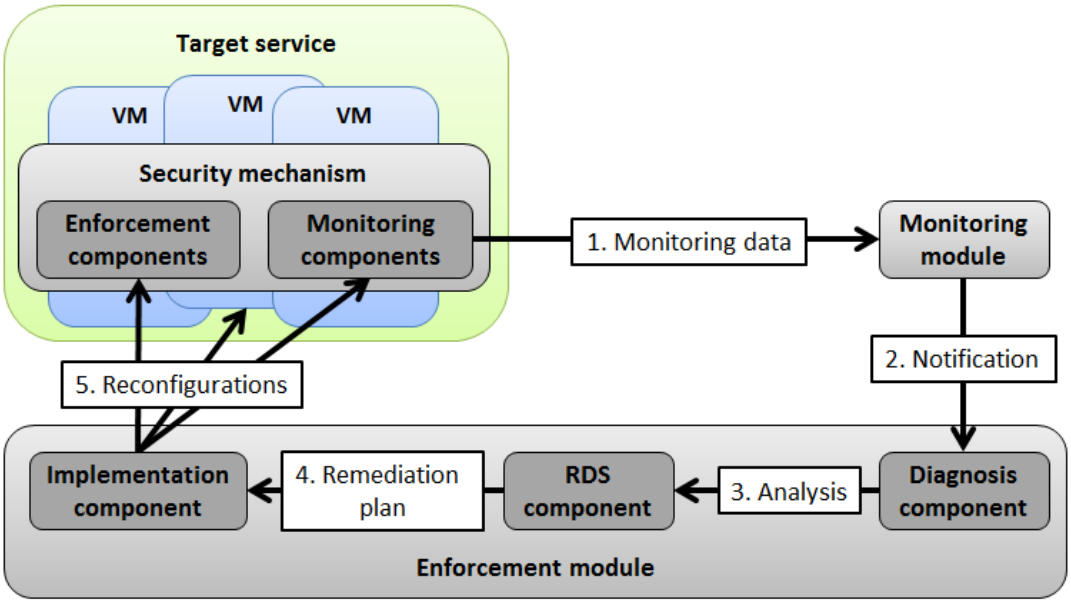


Figure 8. SLA remediation phase

Analysis of monitoring events and identification and execution of remediation plans is further elaborated in the next subsections.

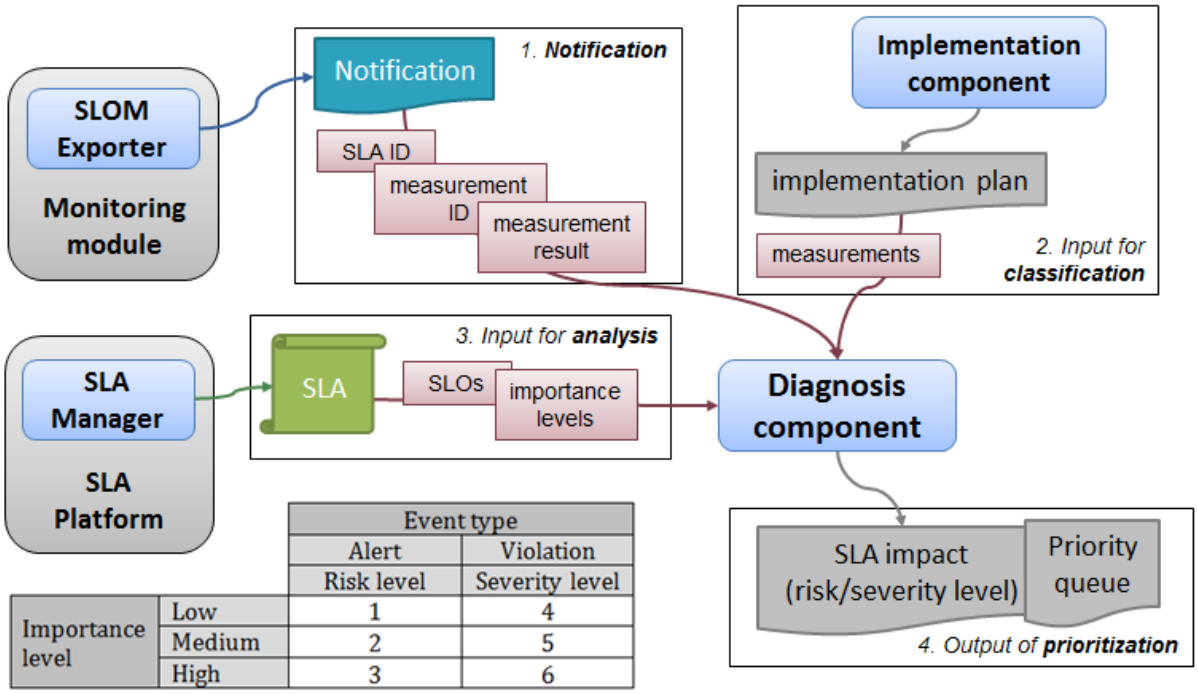
3.2.1. Diagnosis process

During the SLA implementation phase, the metrics in SLOs define the *measurements* that need to be continuously taken, and the metric values in SLOs define the *thresholds* that need to be respected at all times. Whenever the Monitoring module detects a deviation of some measurement (some measurement value is above or below the defined threshold), the Diagnosis component is notified and is responsible for analysis of the event.

The Diagnosis component performs the following steps:

- **Classification**: This step comprises the identification of affected SLOs in order to determine whether the event represents an alert or a violation.
- **Analysis**: The Diagnosis component checks the importance levels of SLOs affected by the alert/violation in order to determine the risk/severity level of the notified event.
- **Prioritization**: The analysed alert/violation I out in the priority que according to the determined risk/severity level.

In particular (see Figure 9), the Diagnosis component uses the information reported with the notification (SLA ID, ID of the deviated measurement, and the measurement result) to identify the affected SLOs. This mapping is done according to the implementation plan for the alerted/violated SLA. When the event type is determined, the Diagnosis component retrieves the affected SLA to extract the importance levels of the affected SLOs. The risk/severity level is determined for each affected SLO separately according to the event type and the importance weight (as shown with the Event type/importance level table in Figure 9). The determined risk/severity levels defined the impact of the event on the entire SLA (we take the maximum risk/severity level of the affected SLOs) and according to that score the SLA is prioritized.



		Event type	
		Alert	Violation
		Risk level	Severity level
Importance level	Low	1	4
	Medium	2	5
	High	3	6

Figure 9. Diagnosis process

Since an attack or a system failure may cause more than one alert/violation of an SLA, the remediation process cannot be executed for more than one alert/violation per SLA at the time (performing several reconfigurations at one time on an infrastructure may cause even more damage). Therefore the Diagnosis component triggers the RDS component for one SLA at the time according to the priority queue. The activities carried out by the RDS component are further elaborated in the next section.

3.2.2. Identification of remediation plans

As introduced in deliverable D4.3.2 and outlined in Section 3.1.2, the metrics and metric values, which define SLOs in an SLA, define the so-called monitoring rules (i.e., measurements that need to be continuously taken and the thresholds that need to be respected). If the conditions of a monitoring rule are not met, this represents a monitoring event. In SPECS, each measurement is associated to one monitoring event and for each monitoring event there exists one remediation plan.

A *remediation plan* is a sequence of remediation action. A *remediation action* either comprises

- a single monitoring action (i.e., taking a measurement) or
- a sequence of one or more enforcement actions (i.e., reconfigurations) followed by a monitoring action.

For example, if at one point one of the deployed web servers is unresponsive, the remediation plan would include restarting the server or deploying a new one if restarting it fails. In terms of remediation actions this would mean:

- **Remediation action 1**:
  - Enforcement action: Restart the web server.
  - Monitoring action: Check if the web server is responsive.
- **Remediation action 2**:
  - Enforcement action 1: Acquire a new VM.
  - Enforcement action 2: Install a new web server.
  - Monitoring action: Check if the web server is responsive.

As shown in Figure 10, in order to identify the remediation plan for an alert/violation, the RDS component uses the information provided by the Diagnosis component in *Step 1* (IDs of the deviated measurement and the affected SLA), the implementation plan associated to the alerted/violated SLA in *Step 2* (to identify the compromised/failed security mechanism) and the metadata of the involved security mechanism in *Step 3* (to extract the remediation plan for the deviated measurement). When the remediation plan is extracted, it is passed to the Implementation component to execute it (*Step 4*). After all performed reconfigurations, the implementation plan associated to the remediated SLA has to be updated and the SLA either re-enters the monitoring phase (if the remediation is successful) or it undergoes the renegotiation process (if we failed to automatically resolve the issue).

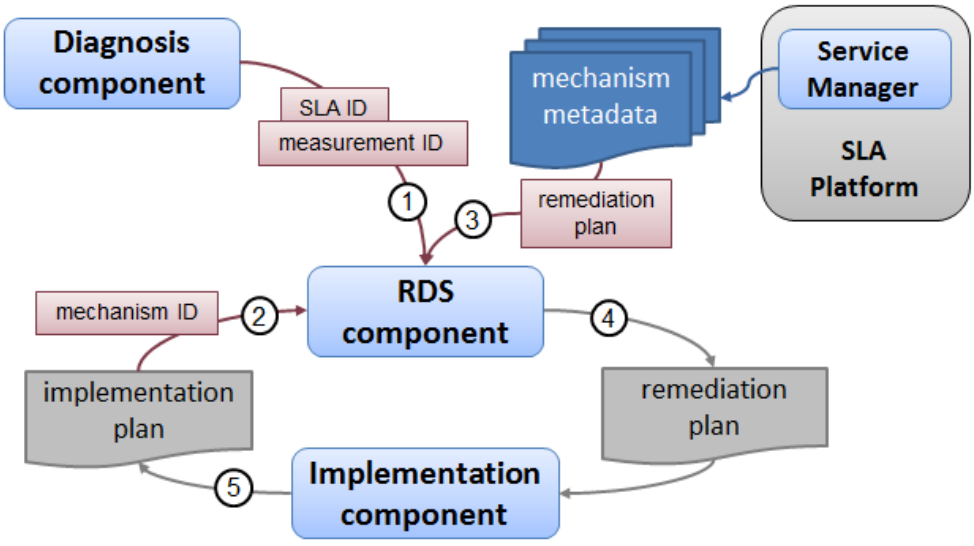


Figure 10. Identification of remediation plans

The details about the execution of remediation plans are provided in the next section.

3.2.3. Execution of remediation plans

When the Diagnosis component analyses an SLA alert/violation and the RDS component identifies the remediation plan that needs to be applied to prevent/recover from it, it is the responsibility of the Implementation component to execute it.

As discussed in Section 3.2.2, a remediation plan consists of a sequence of remediation actions which further comprise a list of enforcement (reconfigurations) and/or monitoring (measurements) actions. Each enforcement and monitoring action is associated to one Chef recipe so that during the execution of the remediation plan the process can be automatized. The process itself, depicted in Figure 11, is simple. The Implementation component takes the first remediation action defined in the remediation plan, and runs the associated list of recipes (which automatize the reconfigurations and invocation of measurements defined with the remediation action). When all the recipes are applied, the Implementation component checks whether the event has been resolved or not. If the event has been resolved, the remediation process is completed and the SLA re-enters the monitoring phase. If the alert/violation persists, the Implementation component proceeds to the next remediation action specified in the plan. The process continues until the event is resolved or until there is no further action to execute. In the latter case, the End-user is offered to renegotiate the conditions of the SLA.

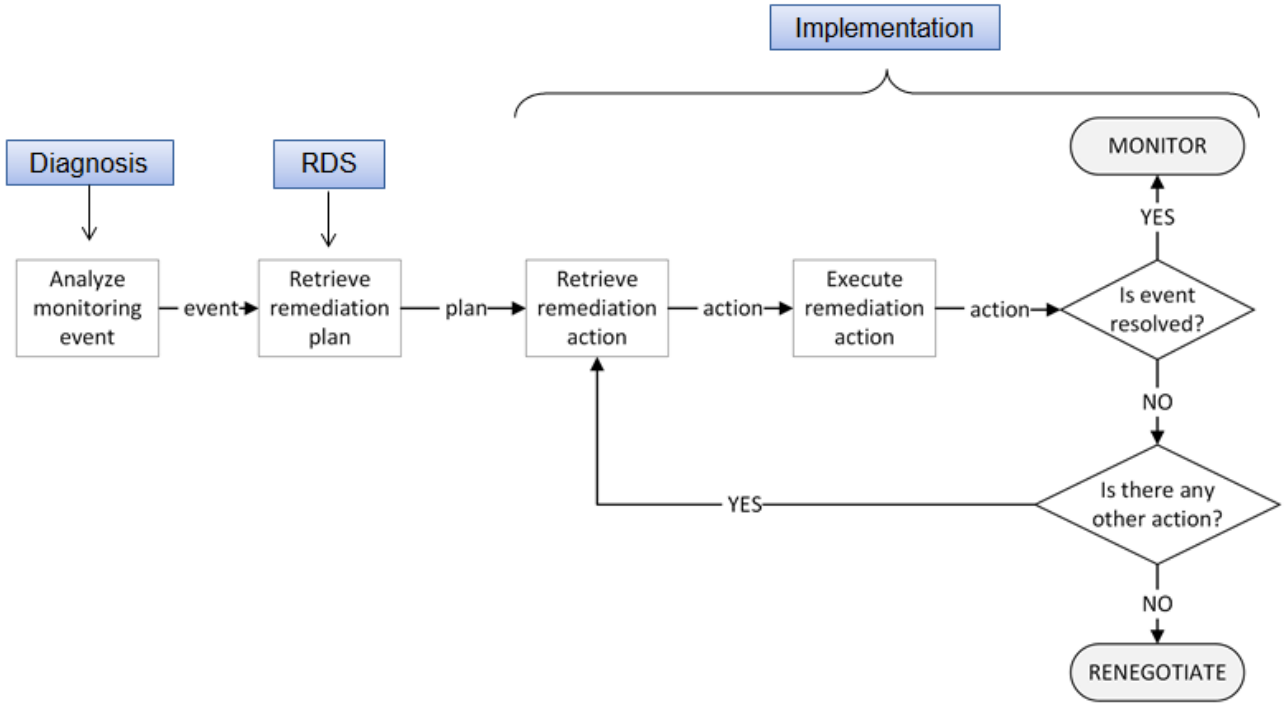


Figure 11. Execution of remediation plans

Since some reconfiguration actions may have required acquisition of new VMs, deployment of new software, etc., the implementation plan for the remediated SLA has to be updated with the relevant new information (IPs, new software, etc.). The Implementation component updates the existing implementation plan and stores it in the Chef Server component.

3.3. Status of implementation activities

In deliverable D4.3.2 we reported the status of implementation activities at M24. Since some requirements have not been implemented at that point, in this section we report about how the remaining requirements associated to the main Enforcement component have been implemented and why/if any are left uncovered.

In Table 1 we present coverage of requirements associated to Enforcement module by main Enforcement components. For the sake of completeness, we report all requirements including the ones already implemented at M24 (highlighted in green).

Requirements for main Enforcement components	SPECS software components				
	Planning	Implementation	Diagnosis	RDS	Broker
ENF_PLAN_R1-R7	x				
ENF_PLAN_R8-R9	x			x	
ENF_PLAN_R10-R12	x				
ENF_IMPL_R1-R8		x			
ENF_IMPL_R9		x			
ENF_IMPL_R10	x				
ENF_DIAG_R1-R6			x		
ENF_DIAG_R7			x		
ENF_DIAG_R8-R18			x		
ENF_REM_R1-R9				x	
SLA_NEG_R30-R31				x	
ENF_BROKER_R1-R5					x

Table 1. SPECS main Enforcement components and related requirements

There are 56 requirements associated to the core components of the Enforcement module (requirements associated to security mechanisms are discussed in Section 4.1) and all of them are implemented with prototypes presented in this document. The requirements that were not implemented with prototypes presented at M24 are mostly related to planning and implementation activities after renegotiation and termination of an SLA.

Requirements *ENF_PLAN_R8* and *ENF_IMPL_R9* are directly associated to building and implementing a reaction plan, which defines actions to be taken after an SLA is renegotiated or terminated. These functionalities are considered covered by the final Enforcement prototypes (see Section 3.4). Requirement *ENF_PLAN_R9*, which is associated to building migration plans, has not been implemented. The last requirement, namely the *ENF_DIAG_R7*, which is associated to the expression of SLA violations in terms of KPIs, is covered by the Diagnosis component since it expresses the effects of a violation in terms of affected SLOs. No improvements or enhancements were required to cover this requirement.

The final prototypes of the Enforcement module are available on Bitbucket:

- Planning: [8]
- Implementation: [9]
- Broker: [10]
- Diagnosis: [11]
- RDS: [12]

In the following section we discuss functionalities of the Enforcement components associated to activities after SLA renegotiation and SLA termination.

3.4. Updated Planning and Implementation components

In deliverable D4.3.2 we presented functionalities of the Planning and the Implementation components associated to the generation of supply chains and the generation and execution of implementation plans associated to a signed SLA. We also presented an initial version of the planning and implementation activities for a renegotiated SLA (generation and execution of so-called reaction plans). In this section, we report the final version of the enforcement steps after the renegotiation/termination and present the updates to the prototype of the Planning and Implementation components.

The updates presented in this section are only associated to functionalities that have been considered as not yet implemented with the previous version of the Enforcement prototypes. Therefore the interested reader is directed to deliverable D4.3.2 for the installation and usage guides, that remain the same, and to deliverable D4.5.3 for the unit tests.

3.4.1. After SLA renegotiation

During the renegotiation phase, the End-user can do the following:

- Change the negotiated metric value for one or more metric already negotiated in the initial SLA (i.e., the End-user can change the threshold in one or more existing SLOs).
- Add or remove one or more security metrics inside a security capability already negotiated in the initial SLA (i.e., the End-user can add or remove one or more SLOs inside an existing security capability).
- Add one or more security metrics under a new capability (i.e., the End-user can add one or more new capabilities previously not included in the SLA).
- Perform a combination of all of the above.

These options can imply the change in the number of VMs, the change in the set of the deployed mechanisms, the change in the allocation of mechanisms' components on the resources, or even reconfigurations of the deployed components. Therefore, the newly signed SLA is accompanied with a new supply chain.

In deliverable D4.3.2 we defined that the Planning component would compare the old and the new SLA and generate a set of "fake violations" (the so-called reaction plan) in order to immediately remediate them and thus align the infrastructure deployed for the initial SLA with the new supply chain. The fake violations would imply a remediation plan that would later be pushed to the Implementation component to execute it.

Every SLA has a validity period. For every renegotiated SLA this period is set to 24h after the signature. This enables us to simplify the implementation of a renegotiation SLA as follows. The Planning component upon receiving the renegotiated SLA simply considers it as a new SLA and repeats the steps defined for a negotiated SLA. Instead of comparing the new supply chain with the old one and generating a list of fake violation, the Planning component retrieves the old implementation plan and updates it with new information implied by the renegotiated SLA (e.g., adds components that need to be newly deployed). In this case, the reaction plan is simply an updated implementation plan. The new implementation plan is

stored with the Chef Server and the Broker component verifies that the infrastructure is compliant to the updated implementation plan (checks whether the list of recipes in the Chef run-list is aligned to the list of recipes specified in the update implementation plan; if anything is missing in the run-list, the Broker adds it). Afterwards the Planning prepares the SLA with alerts and updates the MoniPoli.

The process is depicted in Figure 12. After the End-user signs a renegotiated SLA, the SPECS Application triggers its implementation (it sends a *reconfiguration.json* [38] to the Planning component with an *Update* label in *Step 1*). Afterwards the Planning component stores the reconfiguration file (*Step 2*), logs component's activation (*Step 3*), and identifies and updates the associated planning activity with the reconfiguration ID (*Steps 4-5*). In order to prepare a reaction plan, the Planning component retrieves the new SLA (*Steps 6-7*), the associated new supply chain (*Step 8*), and the old implementation plan (*Steps 9-10*). It builds a reaction plan by making a copy of the old implementation plan and adding information implied by the new supply chain (*Step 11*), stores it in the Chef Server (*Step 12*), and triggers the Implementation component to verify correctness of implementation (*Step 13*). The Broker component then checks if the list of recipes specified in the reaction plan is aligned to the Chef run-list. If anything is missing, it adds it (*Step 14*). Finally, the Planning builds the new SLA with alerts that include the renegotiated rules (*Step 15*), reconfigures the MoniPoli (*Step 16*), updates the SLA state to *Monitoring* (*Step 17*) and logs its deactivation (*Step 18*).

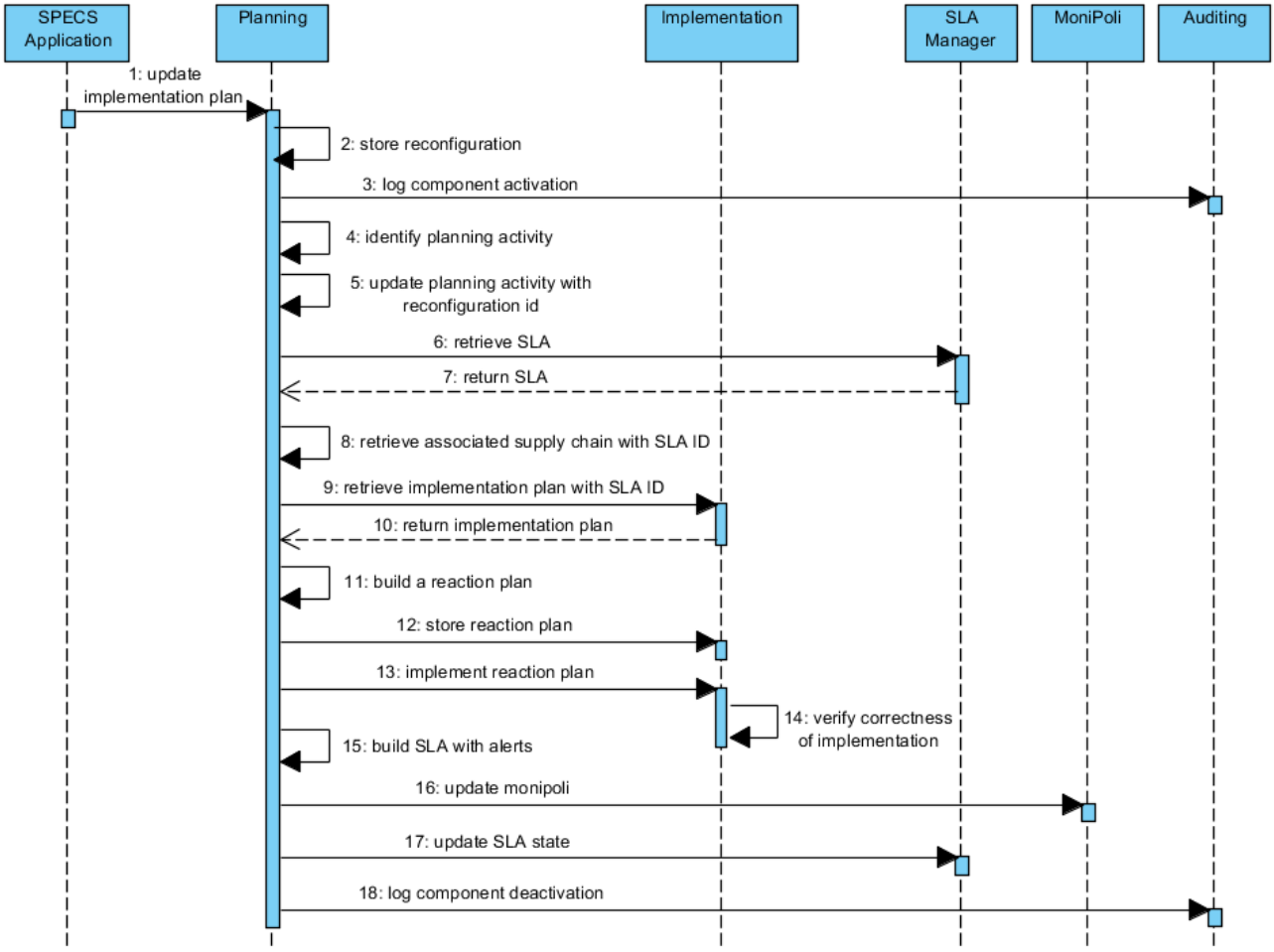


Figure 12. Planning and implementation after SLA renegotiation

In this way, reconfigurations associated to the deployment of new mechanisms on the existing resources are automatically applied by aligning recipes in the Chef run-list. When renegotiated SLA does not require a deployment of a new mechanism but requires reconfigurations of existing ones (in this case Chef recipes are already aligned), we let the infrastructure to adapt itself automatically through the remediation process. In particular, the monitoring adapters deployed on the cloud resources continue to collect the monitoring data and report it to the Monitoring module. Since the MoniPoli holds new rules that are adjusted to the renegotiated SLA, it will create SLA violations which will be automatically remediated. Since these reconfigurations happen during the period where the renegotiated SLA is not yet valid, violations are not notified to the End-user and no remedies have to be applied (no penalties have to be payed and no compensations are due). This process simply allows for the infrastructure to adapt itself to the renegotiated SLA.

With this slight change, the Planning component already covered the majority of these steps after the renegotiation with the prototype presented at month 24 in deliverable D4.3.2 (generation and execution of the implementation plans were implemented at M24).

Let us now consider a few examples.

Example 1 (The change of a metric value requires acquisition of new VMs). Let us assume that an End-user negotiated an SLA for the secure web container service which includes SLOs *Level of Redundancy (LoR) = 3* and *Level of Diversity (LoD) = 2*. SPECS acquires 4 VMs, namely 3 for web servers (to fulfil LoR) which are deployed with 2 different software instances (to fulfil LoD) and 1 VM for a load balancer. Note that both metrics are enforced and monitored with the SPECS WebPool security mechanism (for further details see deliverable D4.3.2).

After some time, the End-user wants to increase the level of redundancy, therefore she/he renegotiates the SLA and sets *LoR = 4*. After the new SLA is signed, the Planning component prepares a reaction plan and stores it in the Chef Server. The reaction plan is the old implementation plan updated with information about a new VM that is required and the components that have to be deployed on top of it. When the Broker checks the list of recipes in the Chef run-list, it detects that one VM is missing. It acquires a new VM and deploys and configures a new web server on top of it to ensure the required level of redundancy.

To update the MoniPoli component, which filters the monitoring data to detect potential or actual SLA violations, the Planning component generates the SLA with alerts, which is based on the new metric values, and reconfigures the MoniPoli.

The renegotiated SLA is now implemented and valid.

Example 2 (The change of a metric value requires reconfiguration of components). An End-user signed an SLA with SLOs *Level of Redundancy (LoR) = 2* and *Scaning Frequency (SF) = 96h*. The first SLO requires an acquisition of 3 VMs for 2 web servers and a load balancer, and is enforced with the SPECS WebPool security mechanism. The second SLO requires periodic vulnerability scans of the deployed web servers, and is enforced with the SPECS SVA security mechanism (for further details about mechanisms see deliverable D4.3.2 and Section 4.3).

After a while, the End-user expresses the need to have more frequent vulnerability scans, therefore she/he renegotiates the SLA and sets *SF = 24h*. After the new SLA is signed, the Planning component prepares a reaction plan (updates the old implementation plan with a new threshold associated to the scanning frequency SLO) and stores it in the Chef Server, and generates a new SLA with alerts to update the MoniPoli.

Since the Chef run-list already includes all recipes (indicating that all components that need to be deployed on the resources in order to fulfil the SLA commitments are already deployed), thus the verification of the correctness of implementation conducted by the Broker component is successful.

In the initial SLA implementation phase, during the deployment of the security mechanisms on acquired resources, the Implementation component sends another set of recipes to the Chef Server (one for each mechanism) that continuously verify that configurations of components installed on the VMs are aligned to the implementation plans. In our example this means that during the initial deployment of the SVA mechanism on the acquired VMs, the Implementation component also uploads the recipe to the Chef Server with which the Chef Server continuously verifies if the SVA components are configured so that the scans are executed every 96h. So, when the Chef Server gets the new implementation plan, it will automatically reconfigure the SVA components and change the frequency of scans to 24h.

Before the reconfiguration takes place (the recipe run list is executed periodically), the SVA monitoring adapter continues to collect the monitoring data (observing the age of the scanning report) and report the measurements to the Monitoring module. Since SPECS initially set up vulnerability scans to be executed every 96h and the updated MoniPoli rule states that the age of the scanning report should not be higher than 24h, the Monitoring module notifies the Enforcement about an SLA violation (the monitoring adaptors at one point reported *SF > 24h*). The Diagnosis component analyses the event and determines that the violation is associated to a failed/missed vulnerability scan. The RDS component afterwards prepares the remediation plan and triggers the Implementation component to execute it. The Implementation component (according to the remediation plan for the SF metric; see D4.3.2) manually triggers a new vulnerability scan to ensure the committed vulnerability scanning frequency.

When the reconfiguration of the scanning frequency is complete, the renegotiated SLA is implemented and valid. Before that time all detected violations fall in the period of the renegotiated SLA not yet being valid, thus they are not notified to the End-user and no remedies are required.

Example 3 (Adding a metric requires replacement of components). Let us assume that an End-user negotiated an SLA with *Level of Redundancy (LoR) = 3*. SPECS implements the SLA by enforcing the WebPool mechanism (acquires 4 VMs, 3 for web servers and 1 for a load balancer).

Further we assume that the End-user later decides to add another security feature to the negotiated service. She/he renegotiates the SLA adding the *Level of Diversity (LoD) = 2* SLO.

This addition requires SPECS to use two different softwares for the deployed web servers instead of one.

After the new SLA is signed, the Planning component prepares a reaction plan (the old implementation plan updated with recipes for the new web server) and a new SLA alerts, and send it to the Chef Server and the MoniPoli, respectively.

When the Broker checks the list of recipes in the Chef run-list, it detects that the recipe for the new web server is missing. It deploys and configures a new web server on top of one of the existing VMs to ensure the required level of diversity. Consequently, the SLA is automatically implemented and valid.

Example 4 (Adding a capability requires installation of new components). In the last example we take an End-user that signs an SLA with SLO *Level of Redundancy (LoR) = 2*. SPECS implements the SLA by enforcing the WebPool mechanism, i.e., by acquiring 3 VMs for 2 web servers and 1 load balancer.

After a while, the End-user expresses the need to security harden the negotiated web servers by adding the Software Vulnerability Assessment capability. The End-user renegotiates the SLA adding the SVA capability and the *Scanning Frequency (SF) = 24h* SLO. This addition requires a set of SVA components to be deployed on the existing VMs hosting the web servers.

The Planning component creates a reaction plan (updates the old implementation plan with information about which SVA components should be deployed on which VMs) and sends it to the Chef Server. When the Broker tries to verify correctness of implementation, it detects that the recipes for the SVA mechanism are missing in the Chef run-list. It adds them and the components are automatically deployed when recipes are run.

Additionally, the Planning generates a new SLA with alerts, which includes rules for the new SVA SLOs and reconfigures the MoniPoli.

After the Broker sets up the SVA mechanism on the initially acquired VMs, the renegotiated SLA is implemented and valid.

3.4.2. After SLA termination

The End-user can decide to terminate the SLA at any given point. Similarly, the End-user can decide to terminate the SLA if at some point some SLA violation occurred and SPECS was not able to remediate it automatically. To this end, the Planning and the Implementation components implement a functionality to terminate an SLA as depicted in Figure 13.

In particular, when the End-user decides to terminate her/his SLA, the SPECS Application triggers the Planning component by sending to it the *reconfiguration.json* [38] with a *Terminate* label (*Step 1*). Afterwards the Planning component stores the reconfiguration file (*Step 2*) and logs component's activation (*Step 3*). Then it triggers the Implementation component to terminate the SLA (*Step 4*) which is orchestrated by the Broker (*Step 5*). Finally, the Planning component triggers the MoniPoli to remove all rules associated to the

terminated SLA (*Step 6*), updates the SLA state to *Terminating* (*Step 7*) and logs its deactivation (*Step 8*).

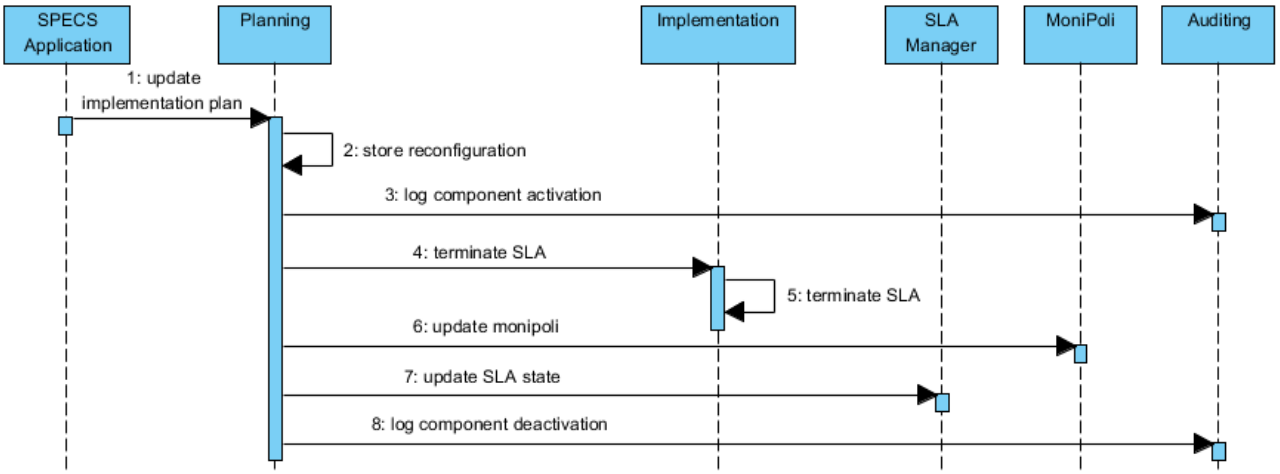


Figure 13. Planning and implementation after SLA termination

4. Security mechanisms

As discussed in deliverable D1.5.1, we have developed a set of SPECS applications with which the End-users can negotiate the preferred or required level of security for a set of cloud services. To enforce and monitor the negotiated security features, we have developed a set of security mechanisms following the user stories and validation scenarios defined in task T5.1.

Each cloud service offered by SPECS is implemented with one mandatory and a set of optional security mechanisms as reported in Table 2.

Security service/ SPECS application	Negotiable security mechanisms			
	Mandatory	Optional		
Secure Web Server	WebPool	SVA	TLS	DoS
Secure Storage	DBB	E2EE		
ngDC	DBB	E2EE		
AAAaaS	AAA	DBB	E2EE	

Table 2. SPECS security mechanisms offered with different SPECS services

The Secure Web Server service provides pools of web servers and assures resilience to security incidents through redundancy and diversity. Further security features can be enforced and monitored with the Software Vulnerability Assessment (SVA) mechanism, the TLS protocol, and the Denial of Service Mitigation and Detection (DoS) mechanism. The WebPool and the TLS mechanisms have been finalized at M24 and presented in deliverable D4.3.2. The initial prototype of the SVA mechanism has been presented in deliverable D4.3.2 at M24; however, in this document (in Section 4.3) we present its final version. In this document (in Section 4.4) we also present the final prototype of the DoS mechanism.

The Database and Backup (DBB) and the End-2-End Encryption (E2EE) mechanisms, which provide storage with backup and client-side encryption, respectively, have been comprehensively discussed in deliverable D4.3.2. Further development of these two mechanisms has been conducted under the task T5.2 (see deliverables D5.2.1 and D5.2.2), which is focused on the development of the Secure Storage application and management of security incidents in the cloud storage domain. The mechanisms are also used with the Next-Generation Data Centers (ngDC) application (presented in deliverable D5.3) and the AAA-as-a-Service application (described in deliverable D5.4). In Section 4.2 we report about the changes/improvements with respect to the initial prototype.

The Authentication, Authorization, and Auditing (AAA) mechanism, which is the base for the AAAaaS application, is fully described in Section 4.5.

Each security mechanism enforces and/or monitors a set of security metrics that End-users can negotiate in an SLA. For this purpose, each security mechanism has a particular architecture that (i) enables enforcement and monitoring of these security metrics and (ii) allows for an automated management. In this document (in Appendix 1) we provide with a comprehensive guide to develop a new security mechanism and add it to the SPECS platform.

4.1. Status of implementation activities

In this section we report about the final status of implementation activities related to SPECS security mechanisms. Although development of some mechanisms was finalized at M24, for the sake of completeness, we discuss the status of implementation for all SPECS mechanisms.

In Table 3 we present the coverage of requirements associated to SPECS security mechanisms. All together we have 32 requirements and the ones highlighted in green have already been covered by prototypes presented in deliverable D4.3.2. The coverage of the remaining 14 requirements we discuss in the following.

Requirements for security mechanisms	SPECS security mechanisms						
	WebPool	DBB	E2EE	SVA	TLS	DoS	AAA
ENF_POOL_R1-R4	x						
ENF_POOL_R5	x						
ENF_TLS_R1-R5					x		
ENF_SVA_R1-R2, R4				x			
ENF_SVA_R3							
ENF_CRYPTO_R1-R4			x				
ENF_AAA_R1-R9							x
ENF_DOS_R1-R3						x	
ENF_DBB_R1-R2		x					

Table 3. SPECS security mechanisms and related requirements

The *ENF_POOL_R5* requires the WebPool mechanism provide incident management functionalities. In particular, the requirement assumes that in the case of security incidents the WebPool mechanism is able to isolate the affected/targeted Virtual Machines (VMs) to ensure business continuity to the End-user. At current state, the requirement has been covered by devising proper remediation actions (see D4.3.2 for a detailed explanation).

As already anticipated in deliverable D4.3.2, the *ENF_SVA_R3*, associated to automated vulnerability patching, remains uncovered due to complexity of the problem. However, in Section 4.3 we present integration of the existing SVA mechanism with the OpenVAS vulnerability scanner, which improves the coverage of requirements *ENF_SVA_R1* and *ENF_SVA_R2*.

We have nine requirements for the AAA mechanisms. Of these, six are covered by the current prototype (*ENF_AAA_R4-R9*), one is deprecated (*ENF_AAA_R1*), and two are not covered (*ENF_AAA_R2-R3*). In particular, *ENF_AAA_R1* involves the use of different external authentication sources. It no longer applies to the AAA mechanism (requirement deprecated) since, as clarified in Section 4.5, the current version of the AAA mechanism includes an OAuth Server and, according to OAuth, the choice of the authentication source is delegated to the client, which is not part of the AAA package that implements the AAA mechanism. As will be discussed later, we do provide an example of OAuth Client along with the AAA package, but it does not include support for external authentication sources. For what concerns the other two requirements, *ENF_AAA_R2* and *ENF_AAA_R3* remain uncovered, even if the OAuth standard includes the support for the management of different accounts.

For what regards the DoS mechanism, its current prototype covers all related requirements. Further details about the mechanism's behaviour and features are given in Section 4.4.

Note that the validation of the Enforcement module is the focus of the task T4.5. Thus further elaboration on how each requirement has been covered is reported in deliverable D4.5.3.

The prototypes of the developed security mechanisms are available on the project' Bitbucket account:

- WebPool: [13], [14]
- TLS: [15]
- SVA: [16], [17], [18], [19], [20]
- E2EE and DBB: [1], [2], [3], [4]
- AAA: [21], [22]
- DoS: [23], [45]

4.2. DBB and E2EE mechanisms

In this section, we present updates with respect to M24 associated to the Database and Backup (DBB) and the End-2-End Encryption (E2EE) security mechanisms.

As introduced in deliverables D4.1.2 (requirements) and D4.2.2 (design), and demonstrated as an initial prototype in deliverable D4.3.2, the DBB and E2EE mechanisms enhance security level of the cloud storage service. They enforce and monitor *confidentiality*, *integrity*, *write-serializability* (i.e., consistency among updates), and *read-freshness* (i.e., requested data always being fresh as of the last update). Moreover, with these two mechanisms, the End-user is provided with proofs of any violations associated to these security features.

The mechanisms were initially developed under the task T4.3 but further improvements were made under the task T5.2, where the focus was on monitoring and automated management of security incidents associated to the cloud storage domain.

In deliverable D4.3.2 we presented the architecture of both mechanisms, introduced security metrics they enforce and monitor, and elaborated on how their violations are automatically remediated. We also provided with the installation and usage guides for the initial prototypes.

In deliverables D5.2.1 and D5.2.2 we provide theoretical and practical, respectively, improvements of the functionalities of these two mechanisms. Namely, we refined the list of associated security metrics, improved the monitoring capability, and increased the performance. Since the focus of this task is automated enforcement and remediation, in the following subsections we only report the relevant changes (namely, the architecture, the refined metrics and associated measurements, and monitoring events and associated remediation plans) and invite the interested reader to see deliverables D5.2.1 and D5.2.2 for any further information.

4.2.1. Updated architecture

The mechanisms comprise a set of components that enforce and monitor security metrics associated to the secure storage service. The Client component locally encrypts and decrypts

data and communicates with the storage servers to upload and download data. The Main DB, Backup DB, Main Server, and Backup Server handle all reads and writes of the End-user's data and orchestrate backups. The Auditor and the Monitoring Adapter component continuously monitor the status of security metrics that both mechanisms enforce.

Apart from the Client component, which performs the encryption/decryption of the data, and the servers and databases which manage the data, the biggest responsibility lies with the Auditor, since it verifies whether the enforced security properties WS, RF, and IN are respected. To this end, we use the so called *attestations* which are signed messages exchanged between the Client and the Main Server components with every `put` or `get` request. Each time the Client performs a `get` (the request contains data block ID and Client's `get attestation`) the Main Server returns the requested data and attaches the Cloud's `cloud get attestation` with which it certifies that the returned data is the right one. Each time the Client performs a `put` (the request contains the data and Client's `put attestation` with which the Client authorizes the overwriting of a certain existing data with a new content), the Main Server stores the data, returns the block ID, and attaches the Cloud's `cloud put attestation` which affirms that the received data is unchanged and successfully stored. The Client automatically forwards a copy of each attestation to the Auditor. After each epoch (i.e., a predefined period of time) or after the Client component triggers it, the Auditor analyses the set of attestations which have to form a correct chain. If any error is detected, this implies a violation of one or more commitments associated to WS, RF and/or IN, therefore the Auditor notifies the Enforcement module about an SLA violation.

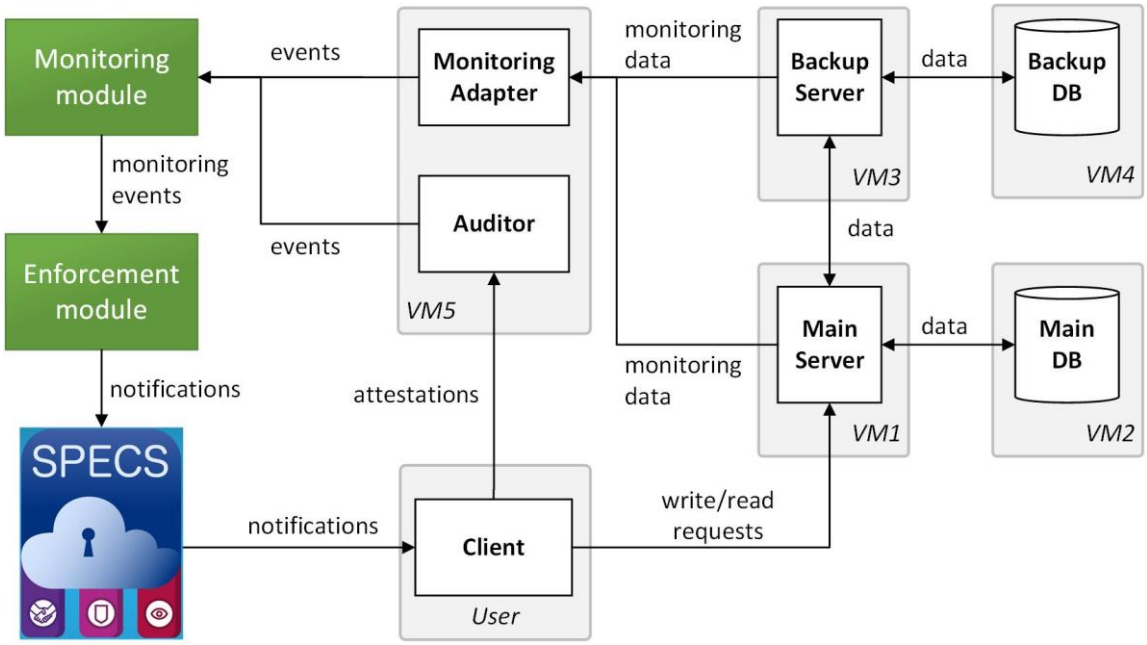


Figure 14. Architecture of E2EE and DBB mechanisms

With respect to M24, the architecture of the mechanisms has been slightly simplified but remains the same functionalities. Instead of two monitoring adapters, the final version of the mechanisms comprises one monitoring adapter taking all measurements previously associated to two adapters. The Client component previously split into two elements (DBB Client and E2EE Client plug-in) is now one and is simply configured according to the security

features selected by the End-user acquiring the secure storage service through SPECS. The updated design is depicted in Figure 14 above.

4.2.2. Updated security metrics and measurements

As already mentioned in the introduction of this section, the E2EE and DBB mechanisms enforce and monitor confidentiality, integrity, write-serializability, and read-freshness of the data stored in the cloud by the End-user. In the following four tables we present the updated security metrics defined for both mechanisms.

Name		Value	Default value	Unit
Write-serializability (**WS**)		yes	yes	n/a
Description	This metric ensures the End-user consistency among updates of the stored data. In case of WS violations, the End-user will be notified and the system will be restored to the state of the last completed backup.			
Actions taken to enforce the metric	1. With every `put` request from the Client the data is stored in the Server's DB. 2. The Server sends `cloud put attestation` to the Client which automatically forwards it to the Auditor. 3. After each *epoch*, the Auditor checks attestation chains. 4. If WS violation is detected, the End-user is notified and the Server's DB is restored to the state of the last finished *epoch*.			

Table 4. DBB security metrics WS

Name		Value	Default value	Unit
Read-freshness (**RF**)		yes	yes	n/a
Description	This metric ensures the End-user that the requested data will always be fresh as of the last update. In case of RF violations, the End-user will be notified and the system will be restored to the state of the last completed backup.			
Actions taken to enforce the metric	1. With every `get` request from the Client the data is retrieved from the Server's DB and sent to the Client. 2. The Server sends `cloud get attestation` to the Client which automatically forwards it to the Auditor. 3. After each *epoch*, the Auditor checks attestation chains. 4. If RF violation is detected, the End-user is notified and the Server's DB is restored to the state of the last finished *epoch*.			

Table 5. DBB security metrics RF

Name		Value	Default value	Unit
Integrity (**IN**)		yes	yes	n/a
Description	This metric ensures the End-user integrity of the stored data.			
Actions taken to enforce the metric	The Auditor continuously (after end of each epoch or when Client triggers it) verifies integrity of the stored data.			

Table 6. DBB security metrics IN

Name		Value	Default value	Unit
Confidentiality (**CO**)		yes	yes	n/a
Description	This metric ensures the End-user confidentiality of the stored data. Confidentiality is enforced with end-2-end encryption provided by the Client component. We guarantee that the Client component is audited and thus (used as is) grants the security of encryption.			
Actions taken to enforce the metric	Before providing the End-user with the link to the E2EE Client component, check if the version of the component is certified (i.e., check if the web store[1] maintains the latest version of the Client).			

Table 7. E2EE security metric CO

With respect to the definitions presented in deliverable D4.3.2, we refined the definitions of metrics WS and RF, now providing stronger security assurance (previous definitions guaranteed that the End-user will be notified about violations of WS and RF, but with updated metrics we ensure that the WS and RF are respected). Furthermore, we added a new security metric IN to the DBB mechanism and renamed and refined the definition of the security metric associated to the E2EE mechanism (the updated metric guarantees confidentiality achieved through the client-side encryption, whereas with the previous version of the metric we assured only that the Client component providing the client-side encryption is certified/audited).

As discussed in Section 3, in order to automate the SLA monitoring phase, each security metric is associated to one or more measurements. Due to the updates made to the set of security metrics, in the following four tables we present the updated measurements associated to them. For each measurement we also report a condition under which the MoniPoli notifies the Enforcement module about a potential SLA alert/violation.

Metric	Write-serializability (WS)
SLO	write_serializability = yes

Measurements	MoniPoli rules
ws_violation	*ws_violation = yes*
fork_attack	*fork_attack = yes*
ws_int_violation	*ws_int_violation = yes*
rollback_attack	*rollback_attack = yes*
system_error	*system_error = yes*
primary_server_availability	*primary_server_availability = no*
primary_db_availability	*primary_db_availability = no*
backup_server_availability	*backup_server_availability = no*
backup_db_availability	*backup_db_availability = no*
backup_restore_completeness	*backup_restore_completeness = no*

Table 8. Measurements and MoniPoli rules associated to DBB metric WS

[1] https://chrome.google.com/webstore/search/e2ee

Metric	Read-freshness (RF)
SLO	read-freshness = yes
Measurements	**MoniPoli rules**
fork_attack	*fork_attack = yes*
different_block_int	*different_block_int = yes*
different_block	*different_block = yes*
primary_server_availability	*primary_server_availability = no*
primary_db_availability	*primary_db_availability = no*
backup_server_availability	*backup_server_availability = no*
backup_db_availability	*backup_db_availability = no*
backup_restore_completeness	*backup_restore_completeness = no*

Table 9. Measurements and MoniPoli rules associated to DBB metric RF

Metric	Integrity (IN)
SLO	integrity = yes
Measurements	**MoniPoli rules**
in_violation	*in_violation = yes*
different_block_int	*different_block_int = yes*
ws_int_violation	*ws_int_violation = yes*
primary_server_availability	*primary_server_availability = no*
primary_db_availability	*primary_db_availability = no*
backup_server_availability	*backup_server_availability = no*
backup_db_availability	*backup_db_availability = no*
backup_restore_completeness	*backup_restore_completeness = no*

Table 10. Measurements and MoniPoli rules associated to DBB metric IN

Metric	Confidentiality (CO)
SLO	confidentiality = yes
Measurements	**MoniPoli rules**
client_code	*client_code = no*

Table 11. Measurements and MoniPoli rules associated to E2EE metric CO

Some measurements are continuously taken by the Monitoring Adapter (the ones associated to the availability of components and completeness of backups/restorations) and some are continuously (after each epoch or after the auditing process is triggered by the Client component) evaluated by the Auditor. In deliverable D5.2.1 we present the auditing process and we refer the interested reader to see the full explanation of the process there, here in Figure 15 we simply outline in blue how the Auditor determines the values of the measurements according to the correctness of the chain of (`get` and `put`) attestations (denoted as CA) collected from the Client component.

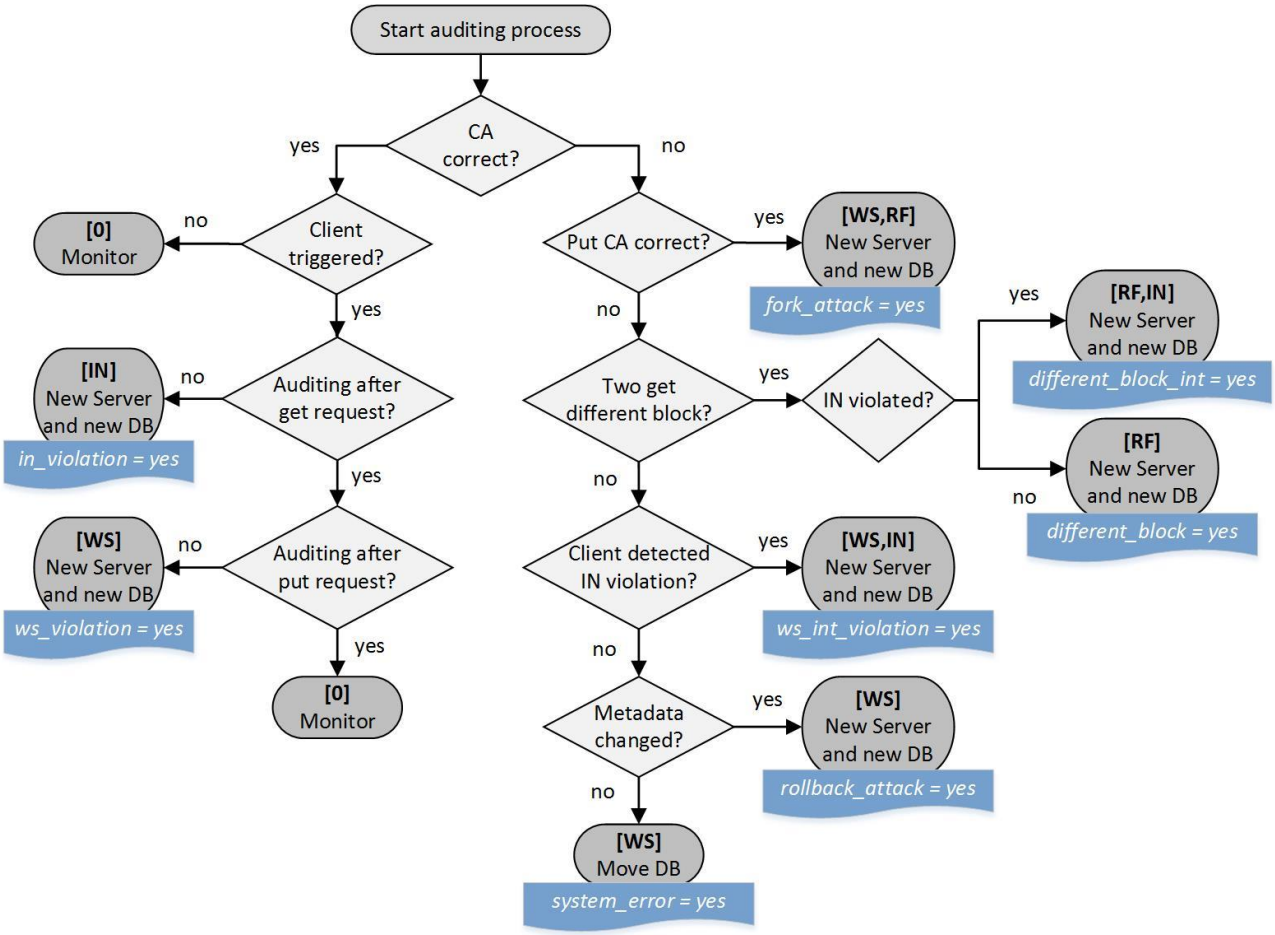


Figure 15. Auditing process and measurements

4.2.1. Updated monitoring events and remediation plans

Each deviation of a measurement is associated to one predefined monitoring event (representing an SLA alert or an SLA violation). In Table 12 we report all monitoring events defined for the updated set of measurements for the E2EE and DBB mechanisms.

ID	Condition	Affected metrics		Event type
E2EE-E1	client_code = no	CO		violation
DBB-E1	in_violation =yes	IN		
DBB-E2	ws_violation = yes	WS		
DBB-E3	fork_attack = yes	WS	RF	
DBB-E4	different_block_int = yes	RF	IN	
DBB-E5	different_block = yes	RF		
DBB-E6	ws_int_violation = yes	WS	IN	
DBB-E7	rollback_attack = yes	WS		
DBB-E8	system_error = yes	WS		
DBB-E9	primary_server_availability = no	WS	RF	alert
DBB-E10	primary_db_availability = no	WS	RF	
DBB-E11	backup_server_availability = no	WS	RF	
DBB-E12	backup_db_availability = no	WS	RF	

Table 12. Monitoring events related to E2EE and DBB metrics

In the next table we report the updated and final set of remediation actions required to mitigate E2EE and DBB alerts and recover from the SLA violations related to E2EE and DBB mechanisms.

ID	Description
E2EE-A1	Upload the latest version of the Client to the web store and check its availability.
DBB-A1	Acquire a new VM in a new pool, set up a new Main Server, connect it to the Main DB, and check if the Main Server is responsive.
DBB-A2	Acquire a new VM in the Main Server pool, set up a new Main DB, and check if the Main DB is responsive.
DBB-A3	Perform restoration (Backup DB to Main DB) and check if it is complete.
DBB-A4	Connect the Main Server to the Backup DB, acquire a new VM in a new pool, set up a new Backup DB, and check if the Backup DB is responsive.
DBB-A5	Restart the Main Server and check if it is responsive.
DBB-A6	Acquire a new VM in the Main Server pool, set up a new Main Server, and check if it is responsive.
DBB-A7	Restart the Main DB and check if it is responsive.
DBB-A8	Restart the Backup Server and check if it is responsive.
DBB-A9	Acquire a new VM in the Backup Server pool, set up a new Backup Server, and check if the Backup Server is responsive.
DBB-A10	Perform backup (Main DB to Backup DB) and check if it is complete.
DBB-A11	Restart the Backup DB and check if it is responsive.
DBB-A12	Acquire a new VM in the Backup Server pool, set up a new Backup DB, and check if the Backup DB is responsive.

Table 13. E2EE and DBB remediation actions

The next two figures present the updated remediation plans associated to alerts and violations of the DBB and E2EE metrics.

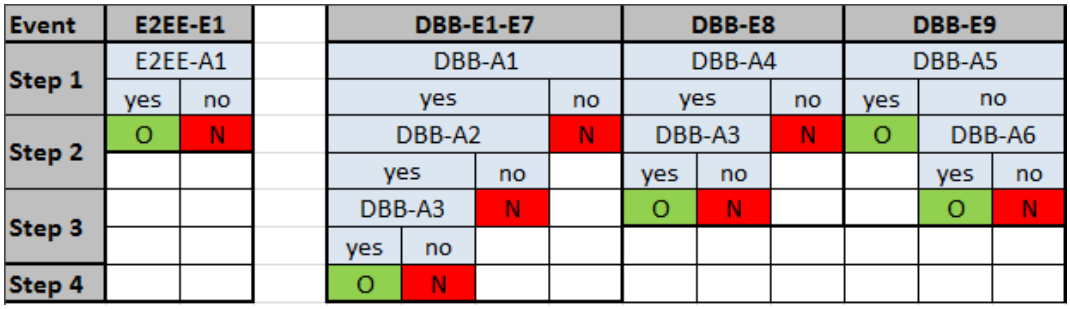


Figure 16. Remediation plans for monitoring events E2EE-E1, and DBB-E1 to DBB-E9

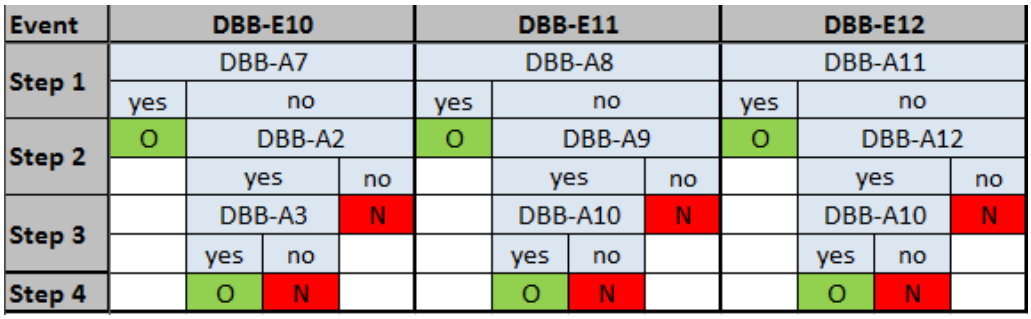


Figure 17. Remediation plans for monitoring events DBB-E10 to DBB-E12

All further implementation and configuration details for both mechanisms are available in deliverables D4.3.2, D5.2.1, D5.2.2 and on project's Bitbucket [5] (in the mechanism metadata format). The code for all DBB and E2EE components is also available on mechanisms' Bitbucket repositories [1], [2], [3], and [4].

4.3. SVA mechanism

The prototype for the SVA security mechanism, which implements the Software Vulnerability Assessment capability, was initially presented in deliverable D4.3.2 (submitted at M24). In this section we focus on functionalities that were defined in the design, but not yet implemented/integrated at M24.

The SVA mechanism comprises a set of components that enhance the security of cloud services with the following functionalities:

- Periodic generation of the list of published software vulnerabilities extracted from different public repositories.
- Periodic vulnerability scans of acquired cloud resources (with a possibility of using more than one scanner).
- Periodic generation of the report outlining available updates and upgrades of vulnerable libraries installed on target services.

These functionalities are offered to the End-user through a set of security metrics:

- **List Update Frequency (LUF)**: With this metric the End-user sets the frequency of updates of the list of published software vulnerabilities. For example, for *LUF = 24h*, SPECS ensures that the list of published software vulnerabilities will be updated and presented to the End-user at least once every 24h.
- **Scanning Frequency – Basic Scan (BSF)**: With this metric the End-user sets the frequency of basic software vulnerability scans (scans that are executed with a single vulnerability scanner). For example, for *BSF = 24h*, SPECS ensures that software vulnerability scans will be performed at least once every 24h.
- **Scanning Frequency – Extended Scan (ESF)**: With this metric the End-user sets the frequency of extended software vulnerability scans (scans that are executed with two different vulnerability scanners). For example, for *ESF = 48h*, SPECS ensures that software vulnerability scans will be performed with two different vulnerability scanners at least once every two days.
- **Up Report Frequency (URF)**: With this metric the End-user sets the frequency of checks for updates and upgrades of vulnerable installed libraries. At the SLA implementation phase, SPECS generates a vulnerability list, performs a vulnerability scan of the cloud resources, and then periodically checks for available updates and upgrades of vulnerable libraries. For example, for *URF = 24h*, SPECS ensures that checks for updates and upgrades are performed at least once every day.
- **Penetration Testing Activated (PTA)**: With this metric the End-user expresses the need for the penetration testing activity. If chosen, a scanner with penetration testing functionality is deployed.

In deliverable D4.3.2 we described the architecture of the mechanism that enforces and monitors three defined security metrics, namely LUF, BSF, and URF. The architecture (depicted in Figure 18) comprised the following components:

- **SVA Enforcement**: This component enforces SVA security metrics; it manages (generates and updates) vulnerability lists, orchestrates scans, checks for updates/upgrades of vulnerable libraries installed on the End-user's target services, and builds reports. For the secure web container cloud service, the SVA Enforcement component is deployed on every VM hosting a web server.
- **SVA Monitoring**: This component monitors security metrics; it monitors all parameters associated to each metric (e.g., age of reports, availability of repositories, and responsiveness of scanners). For the secure web container cloud service, the SVA Monitoring component is deployed on every VM hosting a web server.
- **SVA Scanner**: Executes vulnerability scans. For the secure web container cloud service, the SVA Scanner component is deployed on every VM hosting a web server.
- **SVA Dashboard**: This component presents to the End-user the vulnerability list and scanning results, and reports about available updates/upgrades of vulnerable libraries. For the secure web container cloud service, the SVA Dashboard component is deployed on the VM hosting the load balancer.

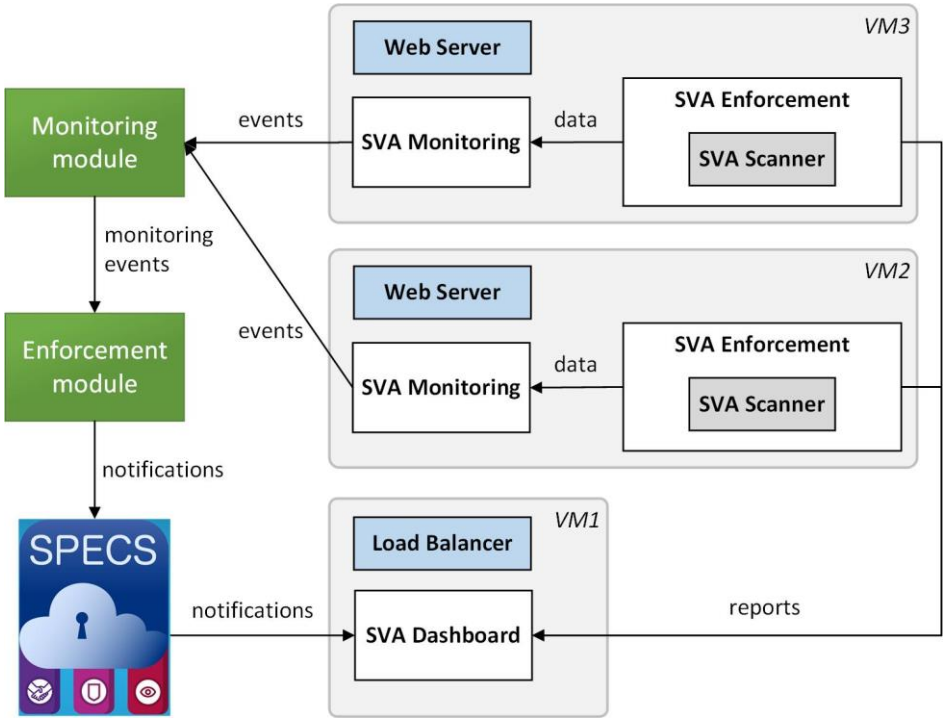


Figure 18. Architecture of the initial SVA prototype

The remaining two security metrics, namely ESF and PTA, require an additional scanner with the penetration testing functionality. In the next subsection we present the integration of such a scanner, namely the OpenVAS introduced in deliverable D3.4.1. We present updates on the SVA architecture, whereas the installation and the usage guides for new components are provided in deliverable D3.4.1.

4.3.1. Updated architecture

In order to enforce extended software vulnerability scans which even provide penetration testing functionality, during the SLA implementation phase SPECS has to deploy an additional scanner on the acquired resources, namely the OpenVAS. As introduced in deliverables D3.3 and D3.4.1, OpenVAS comprises the following components:

- **OpenVAS Scanner**: Executes vulnerability scans.
- **OpenVAS Manager**: Controls the OpenVAS vulnerability scanner and generates scanning reports.
- **OpenVAS Client**: Drives the OpenVAS Manager component by configuring vulnerability scans.

If the End-user negotiates extended vulnerability scans or expresses the need for the penetration testing functionality in the software vulnerability assessment capability, SPECS deploys OpenVAS as depicted in Figure 19.

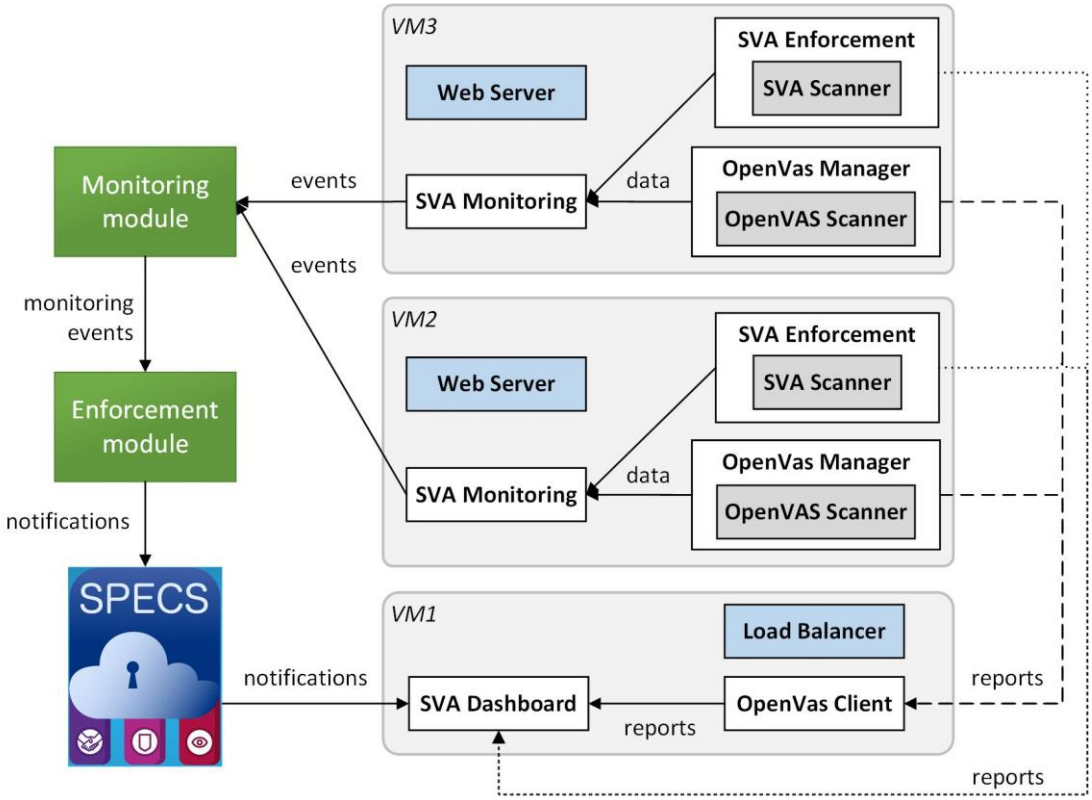


Figure 19. Architecture of the final SVA prototype

For further details about the OpenVAS see deliverables D3.3 and D3.4.1, and for details about the SVA mechanism see deliverable D4.3.2.

4.4. DoS Detection and Mitigation mechanism

In this section, we present the description and the implementation details for the DoS Detection and Mitigation mechanism, designed to offer a solution to identify possible Denial of Service attacks and to automatically mitigate them. The SPECS DoS Detection and Mitigation mechanism's preliminary design was presented in deliverable D4.2.2. In the following, we

provide a comprehensive overview of the architecture, the security metrics associated and the related remediation actions, in addition to the installation and usage guide.

4.4.1. Overview

The DoS Detection and Mitigation mechanism is a negotiable security mechanism offered to End-users. The mechanism has been devised to provide a feature for enriching the Secure Web provision mechanism, discussed in deliverable D4.3.2, in order to protect the delivered web servers against common security attacks. More in general, the mechanism provides a solution to protect a generic web application, even unsecure, deployed on top of a web pool, against DoS attacks.

It is worth noticing that the activation of the mechanism does not grant to the End-user that any DoS attack are identified and mitigated, but that all the correct countermeasures are being correctly applied.

The DoS Detection and Mitigation mechanism relies upon OSSEC [46], which has been already analysed in the context of WP3 (see D3.3), for what regards the monitoring functionalities. OSSEC works on a Log Analysis basis: it continuously parses and analyses the log files produced by the web servers, identifying the events that may reveal a possible security alert or violation.

Figure 20 briefly summarizes how the attack detection process works: the IDS decodes each event (row in the log files) and compares it with a set of fixed rules. When the rules match the event (or the sequence of events) it generates an alert, stored in a DB and/or starting an active response, like the banning of an IP address or closing a specific TCP connection.

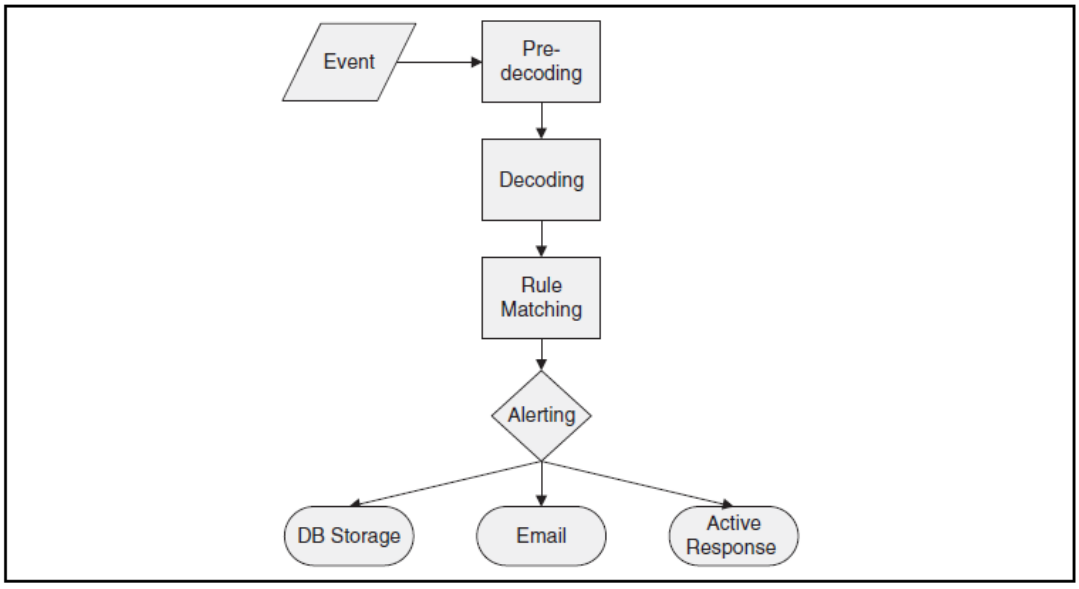


Figure 20. OSSEC execution flow

In order to implement our DoS Detection and Mitigation mechanisms, we customized the open source software, in order to enable its execution inside specs, through ad-hoc Chef

recipes. Moreover, we defined a set of additional rules to protect web application against a set of known threats.

4.4.1.1. Architecture

The DoS Detection and Mitigation mechanism is implemented through the architecture depicted in Figure 21. The main components are:

- **OSSEC Server**: The core of the DoS Detection and Mitigation mechanism, which collects data from the agents and generates alerts and OSSEC actions, in order to reconfigure the web applications.
- **OSSEC Agent**: Resides on the same machine of the monitored web server (in our case a component belonging to the WebPool mechanism), builds the log files and shares the events with the server.
- **OSSEC Adapter**: Collects alerts from the OSSEC Server, generates the reports, and sends the event, in SPECS Event format, to the SPECS Monitoring core (through the Monitoring Core Interface).

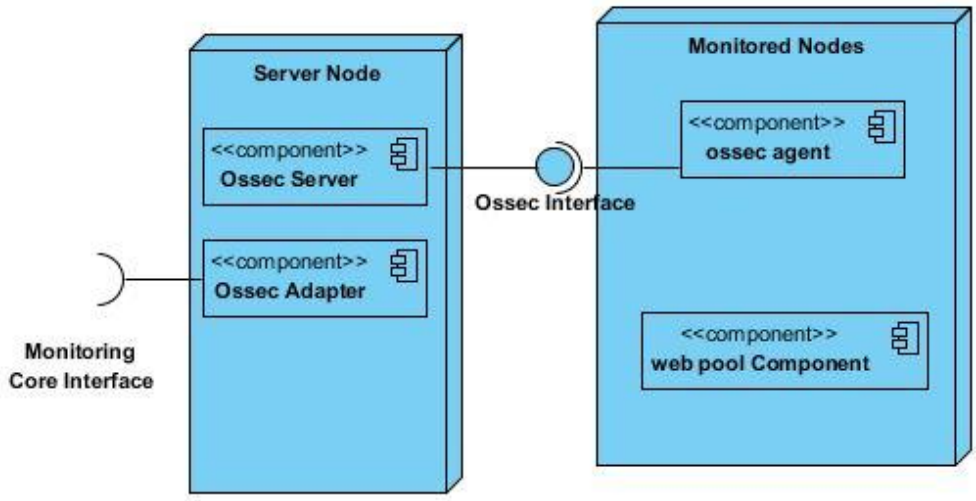


Figure 21. DoS Detection and Mitigation mechanism' architecture

In order to prepare the proper OSSEC configuration to protect the web servers, we adopted a simple continuous process based on the execution of automated penetration tests and on the generation of corresponding protection rules.

In particular, in order to verify the security level of the web containers (and the deployed application on top of it), we adopted a tool offered by OWASP (Zed Attack Proxy Project - ZAP) [47], which performs web-specific penetration tests. In particular, ZAP is an active open source project that continuously updates the set of penetration tests according to known web attacks.

We made a set of tests on different local environments, searching for optimal web-server configuration (modifying accordingly the web-pool servers). Moreover, we enriched the default set of OSSEC rules with additional rules from literature analysis, out of purpose in this deliverable, and through ad-hoc development.

As shown in Figure 22, the tests were made before on local environment (independently from the SPECS deployment), and subsequently by deploying the web containers through SPECS and by testing them accordingly.

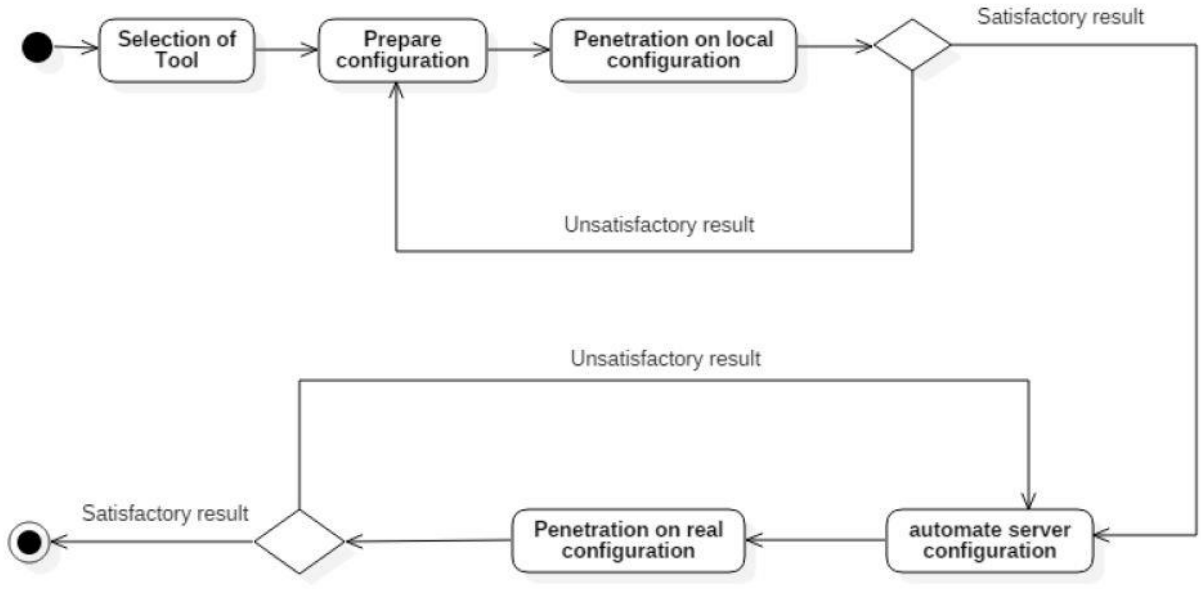


Figure 22. OSSEC Configuration Process

4.4.1.2. Security metrics and controls

One security metric has been associated to the DoS Detection and Mitigation mechanism, and it is defined in the following table, where we provide a description, possible values with units, default values, and actions that need to be taken in order to enforce the metric.

Name		Value	Default value	Unit
dDoS Attack Detection Scan Frequency (**dDoSSF**)		int > 0	24	hours
Description	The frequency of the dDoS attack report generation (for example, "2h" requires that the dDoS attack report is generated every two hours).			
Actions taken to enforce the metric	Before the first check, a dDoS scanning report is generated. Then periodically: 1. Check for updates/upgrades. 2. Build and present report			

Table 14. DoS security metric dDoSSF

The metric has been associated with a basic measurement, reported in the following table along with the MoniPoli rules associated.

Metric	dDoS Attack Detection Scan Frequency (**dDoSSF**)
SLO	dDos_scan_frequency = N hours
Measurements	MoniPoli rules
dDoS_report_age	*dDoS_report_age ≤ N*
dDoS_report_availability	*dDoS_report_availability = yes*

Table 15. Measurements and MoniPoli rules associated to DoS metric dDoSSF

The DoS metric defined above can be associated with the NIST and CCM security controls presented in the following table.

Control Family/Group	Control Name	Control ID	Security metric dDoSSF
NIST			
Access Control	DENIAL OF SERVICE PROTECTION	SC-5	✓
	DENIAL OF SERVICE PROTECTION - DETECTION / MONITORING	SC-5(3)	✓
	CONTINUOUS MONITORING	CA-7	✓
	INFORMATION SYSTEM MONITORING	SI-4	✓
CCM			
Infrastructure and Virtualization Security	Network Architecture	IVS-13	✓

Table 16. Mapping of the DoS metric to NIST and CCM security controls

4.4.1.3. Remediation

As introduced in deliverable D4.3.2 and summarized in Section 3.2, each measurement related to a security metric defines one monitoring event. The following table lists all possible monitoring events related to the DoS metric that can be detected by the Monitoring module.

ID	Condition	Affected metrics	Event type
DoS-E1	dDoS_report_age > dDoSSF_value	dDoSSF	violation
DoS-E2	dDoS_report_availability = no	dDoSSF	alert

Table 17. Monitoring events related to DoS metrics

Table 18 presents actions needed to remediate DoS alerts and violations.

ID	Description
DoS-A1	Delete old scanning report, scan again, and check if the new scanning report is available.

Table 18. DoS remediation actions

The following table presents remediation plan for managing alerts and violations of DoS metrics. For details on the structure of a remediation plan and the remediation process see deliverable D4.3.2 or Section 3.2.

Event	**DoS-E1 (V) / DoS-E2 (A)**	
Step 1	DoS-A1	
	yes	no
Step 2	O	N

Table 19. Remediation plan for alerts and violations related to DoS metrics

4.4.1.4. Development

The development of the DoS mechanism started in the context of WP3 activities, whose result was a preliminary architecture illustrated in deliverable D3.3. Although the associated requirements were already covered, the updated version includes advanced mitigation features. Moreover, in this deliverable we addressed the remediation aspects, not analysed before.

4.4.2. Repository

The previous version of the mechanism is available at [48]. It has been substituted by the two repositories [45] and [23], which respectively include the recipes for the installation of the monitoring adapter and of the enforcement-related components, namely the server and the agents.

4.4.3. Installation

As said, the DoS Detection and Mitigation mechanism is based on OSSEC, which must be properly installed and configured through Chef recipes. As usual, this process requires to have a Chef Server installed and properly configured, and a Chef Workstation from which it is possible to execute the bootstrap of each target node.

The cookbook used for the installation of the OSSEC Server and the OSSEC Agents is available at [23], which also contain the configuration files for OSSEC.

The commands to run on the workstation to install respectively the OSSEC Server and the OSSEC Agent are the following:

```
knife bootstrap <public_ip_address_of_the_node_that_will_hosts_aaa> -x <chef_user_name> -P <chef_user_password> --node-name <node_name> --run-list 'recipe[‘specs-enforcement-ossec::server’]'
```

```
knife bootstrap <public_ip_address_of_the_node_that_will_hosts_aaa> -x <chef_user_name> -P <chef_user_password> --node-name <node_name> --run-list 'recipe[‘specs-enforcement-ossec::agent’]'
```

As shown in Section 4.4.1, the server node must include also an adapter for the Monitoring module. The recipe for the installation of the OSSEC Adapter is available at [45].

4.4.4. Usage

The OSSEC mechanism does not have a web interface enabled at current state, but it is possible to access the information on the alerts that have been generated during its operation by accessing the log file at the path:

```
<public_ip_address_of_the_node_that_will_hosts_OSSEC_Server>/opt/ossec/logs/alerts/alerts.log
```

4.5. AAA mechanism

In this section, we present the description and the implementation details for the AAA mechanism, designed to offer identity management and access control functionalities as-a-

service. The AAA mechanism is offered by the *SPECS AAA package,* involved in the following cross-cutting validation scenarios (reported in deliverable D5.1.2):

- *CRO.5 – User_Direct_Registration*
- *CRO.6 – User_Registration_External_Account*
- *CRO.7 – User_Authentication_External_Account*

The AAA package is also involved in the validation scenarios defined in D5.4 for the *AAA-as-a-Service* (AAAaaS) *validation application:*

- *AAA.1 – Identity_Management_Set-up*
- *AAA.2 – User_Registration*
- *AAA.3 – User_Access_Internal_Account*
- *AAA.4 – User_Access_External_Account*

The SPECS AAA component's preliminary design was presented in deliverable D4.2.2. In the initial design, the AAA mechanism was intended to be used both to manage the identity of the users of the SPECS Platform, and to provide Identity-as-a-service (IDaaS) to generic customers. At current state, the AAA mechanism is devoted to providing access control and identity management features over existing cloud services in an as-a-service fashion, while the management (authentication and authorization) of the SPECS Platform users is provided by the User Manager component, belonging to the SPECS Vertical Layer (see deliverables D1.4.1 and D1.4.2 for related design and implementation details). From the implementation point of view, however, the AAA mechanism uses the same code base with different customizations, minimizing in any case the development effort.

In the following, the behaviour and design of the AAA mechanism is illustrated.

4.5.1. Overview

As anticipated, the AAA mechanism is a negotiable security mechanism offered to the End-users. It enables who acquires them to easily set-up a system for the management of user profile information and for the enforcement of access control policies, requiring only limited skills with authentication and authorization.

In particular, the SPECS AAA mechanism relies upon a well-known open standard for authorization, namely the *OAuth authorization framework* [39]. OAuth provides a "secure delegated access" to protected resources on behalf of the resources' owner, by enabling third-party clients to obtain an access token used in place of the resources' owner's credentials.

Nowadays, OAuth is widely used as a way for Internet users to log into third-party websites using their existing accounts at, for example, Microsoft, Google, Facebook or Twitter, without exposing their password. In this case, Microsoft, Google, Facebook and Twitter act as authorization servers that issue access tokens to (registered) third-party applications, with which these applications can access private resources, i.e., the user profile information. In the case of Twitter, for example, a third-party application registered with the Twitter authorization server may ask for an access token in order to access, for example, the last 5 tweets of a certain user.

The SPECS AAA mechanism provides the End-user with the tools for the management of the users of his/her own applications, including authentication and authorization features enabled through the integration of the OAuth framework. In particular, the AAA mechanism provides a means to allow registered web clients to access information on user profile with the authorization of the profile owners, by means of an access token issued by the OAuth Server. In our implementation, user profile information is stored in an LDAP [40] directory service.

Moreover, the AAA mechanism provides the tools to support a fine-grained role-based authorization thanks to the integration of an XACML-based [41] web authorization system.

4.5.1.1. Architecture

The AAA mechanism is implemented by the AAA package (as shown in Figure 23) and consisting of the following main components:

- The **OAuth Server**, which is the core of the AAA mechanism and includes an **Authorization Server**, used to authorize access requests from external OAuth clients, and the **Resource Server**, representing the entity that manages the resources to protect. The OAuth Server interacts with the **OAuth Client**, which must be included in the web client that intends to use the protected resources (Section 4.5.4 illustrates how to integrate the OAuth Client within a generic target application).
- The **App DB**, where the information of registered OAuth Clients are stored (e.g., the client type, the redirection URI, etc.).
- The **Authentication Backend**, which is used by the OAuth Server to perform client authentication. It includes an **LDAP Authentication Provider** that is connected to an **LDAP Client**, communicating with the **LDAP Server**.
- The **Authorization Service**, which is used by the OAuth Server to perform authorization based on existing XACML policies, and which includes a Policy Enforcement Point (**PEP**), a Policy Decision Point (**PDP**), and a **Policy Repo** to store the policies.

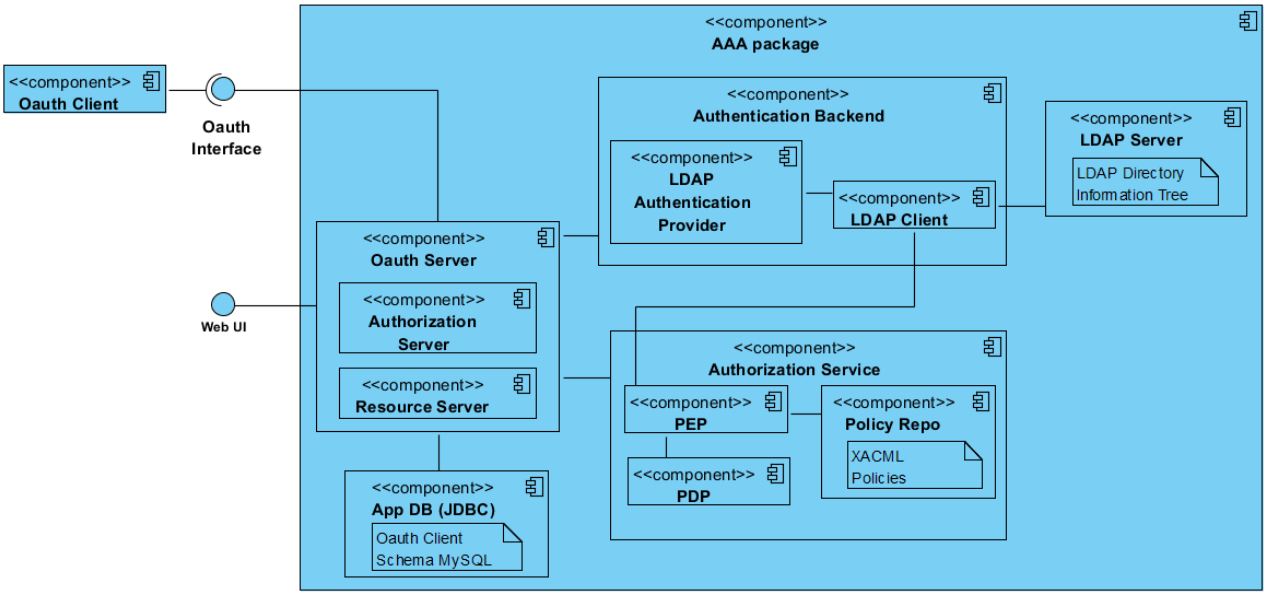


Figure 23. AAA mechanism architecture

The OAuth Server provides an *OAuth Interface* for the communication with the OAuth Client and a *Web Interface* to register new users in the LDAP Server and new applications in the App DB.

In a typical interaction scenario between the OAuth Client and the AAA package, a client application requests the access to a resource of a target application. The OAuth Client forwards the request to the AAA package in order to obtain an *access token*, which enables the Client to access the protected resources on behalf of the resource owner.

The sequence diagram in Figure 24 shows the main interaction among the AAA package components in the presence of an access request from a user.

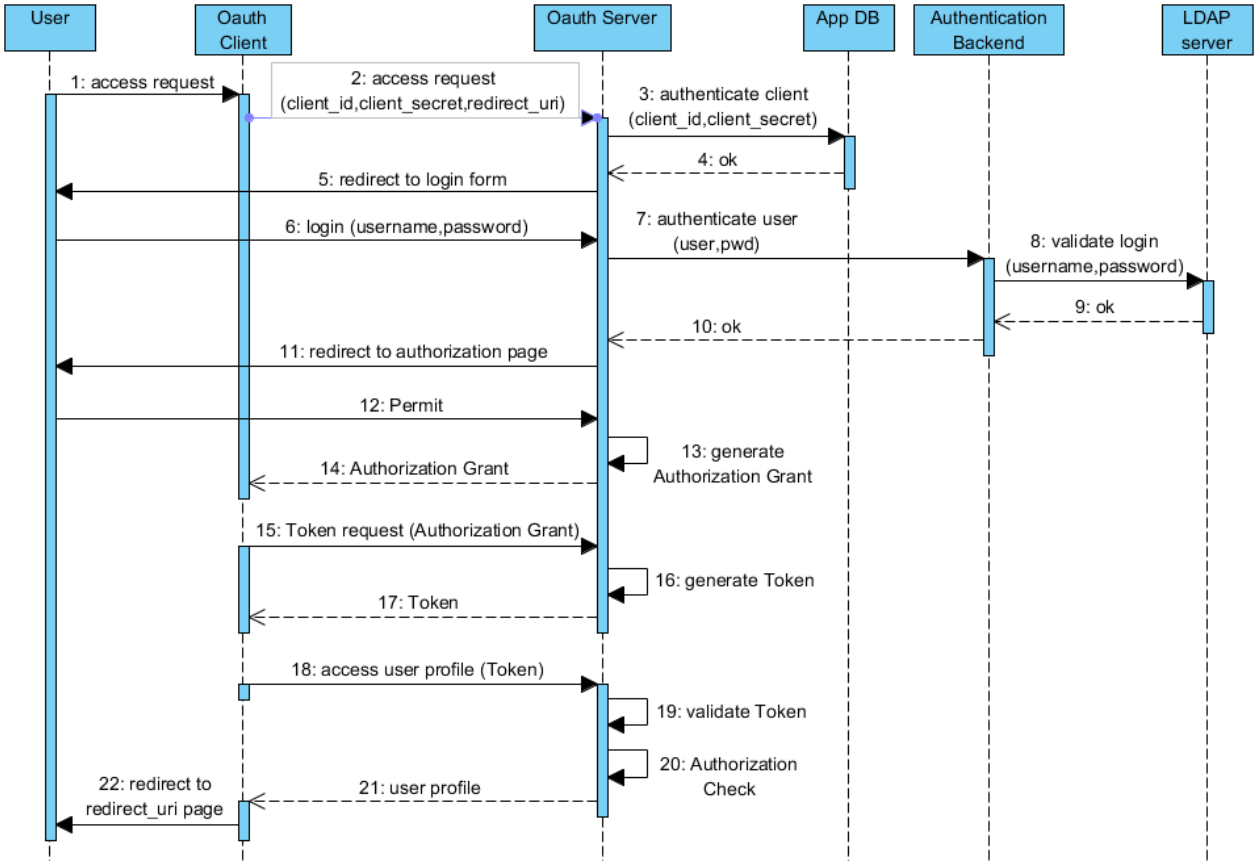


Figure 24. AAA package behaviour in the presence of an access request

By assuming that both the user requesting the access and the used OAuth Client have been previously registered by the OAuth Server of the AAA package, the steps carried out when an access request is issued can be summarized as follows:

1. The OAuth Client, on behalf of the user, submits an access request to the OAuth Server of the AAA package, by providing its id and a redirect URI (*Steps 1-2*).
2. The OAuth Server accesses the App DB looking for the application id; since the application client is registered, the id is found and a login form is presented to the user behind the Client (*Steps 3-5*).
3. The user submits his username and password (*Step 6*).

4. The OAuth Server invokes the Authentication Backend to perform LDAP-based authentication. The user is authenticated and her/his role is retrieved (*Steps 7-9*).
5. The OAuth Server requests to the user the consent to give to the Client the access to protected resources. The user accepts (*Steps 11-12*).
6. The OAuth server generates an *authorization grant* and sends it to Client, which is now allowed to access the protected resources on behalf of the user (note that the actual role-based authorization has not yet taken place) (*Steps 13-14*).
7. The OAuth Client issues a new request to the OAuth Server to obtain a *token*. The request carries the authorization grant and a new Redirect URI. The OAuth Server checks the correspondence between the authorization grant and existing requests and generates an access token, including an expiration time, returned to the Client (*Steps 15-17*).
8. The OAuth Client issues an access request to the Resource Server in the OAuth Server, by using the token. The token includes information on the application and on the role of the user. If a rule is present in the Resource Server for the resources, roles and scope of the request, the Resource Server will grant the access (*Steps 18-22*).

Note that, by default, the Resource Server included in the AAA package manages two basic user roles, namely "user" and "admin", with built-in authorization rules. If the request cannot be evaluated by the Resource Server because the role is different from these two or because the requested resource/scope are not directly managed, the access request is forwarded to the Authorization Service, which makes a decision based on the stored XACML policies. Note that the writing of the policy is under the control of the End-user. Once the authorization is given, the user is redirected to the requested resource. The related flow is depicted in Figure 25.

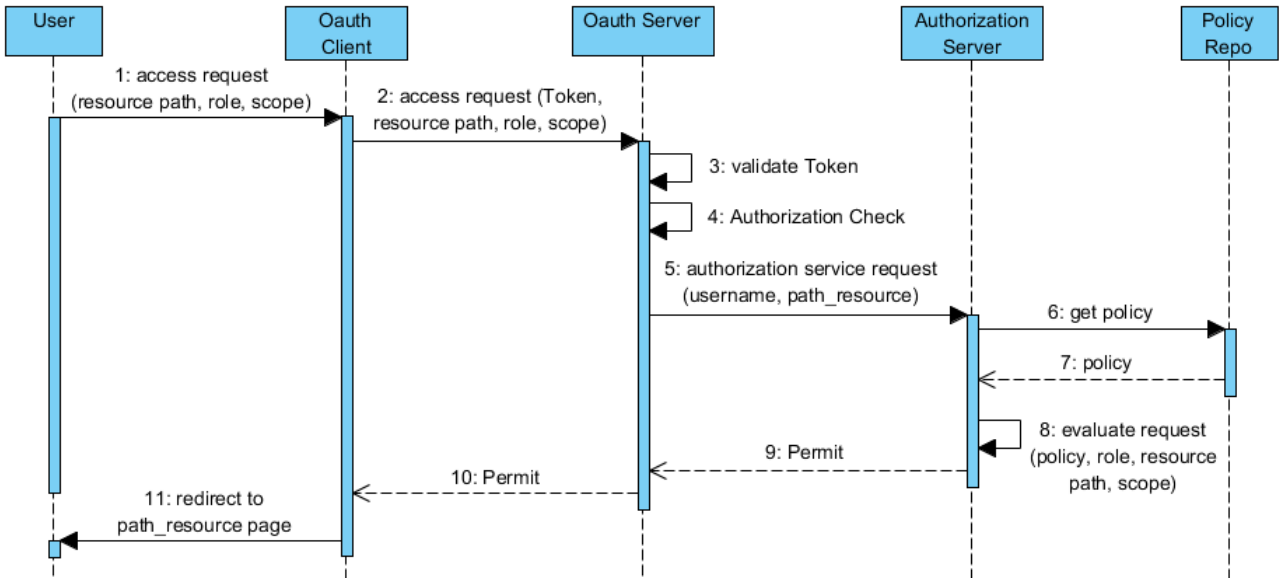


Figure 25. AAA package behaviour in presence of an authorization request managed by the Authorization Service

4.5.1.2. Security metrics and controls

Security metrics associated to the AAA mechanism are defined in the following tables. For each metric, we provide a description, possible values with units, default values, and actions that need to be taken in order to enforce the metric.

Name	Value	Default value	Unit
Secure delegated access (**SDA**)	yes	yes	n/a
Description	This metric ensures that an OAuth Server is configured to ensure authentication and authorization of users and secure delegated access to the users' resources to registered clients.		
Actions taken to enforce the metric	An OAuth server is installed on the End-user's resources to provide secure delegated access to protected resources.		

Table 20. AAA security metric Secure Delegated Access

Name	Value	Default value	Unit
Access report generation frequency (**ARGF**)	int>0	24	hours
Description	This metric sets the frequency of access reports generation. For example, for *access_report_gen_frequency=12*, SPECS ensures that a report is generated at least once every 12 hours.		
Actions taken to enforce the metric	The AAA package is configured so to periodically generate an access report with the requested frequency		

Table 21. AAA security metric Access Report Generation Frequency

Name	Value	Default value	Unit
AAA Log Completeness (**ALC**)	Low/Medium/High	Medium (M)	n/a
Description	This metric represents how detailed the access reports must be.		
Actions taken to enforce the metric	The AAA package is configured so to generate an access report with the requested level of detail.		

Table 22. AAA security metric AAA Log Completeness

As described in deliverable D4.3.2, we associate each metric with a basic measurement and one or more additional measurements (with which the alert/violation thresholds are set and MoniPoli rules are built). The following tables present these measurements together with MoniPoli rules associated to AAA metrics.

Metric	Secure Delegated Access (**SDA**)
SLO	secure_delegated_access = yes
Measurements	**MoniPoli rules**
oauth_server_availability	*oauth_server_availability = yes*
ldap_server_availability	*ldap_server_availability = yes*

Table 23. Measurements and MoniPoli rules associated to AAA metric SDA

Metric	Access Report Generation Frequency (**ARGF**)
SLO	access_report_gen_frequency = N hours
Measurements	MoniPoli rules
access_report_age	*access_report_age <= N*

Table 24. Measurements and MoniPoli rules associated to AAA metric ARGF

Metric	AAA Log Completeness (**ALC**)
SLO	aaa_log_completeness = L/M/H
Measurements	MoniPoli rules
log_detail	*log_detail = ALC_value*

Table 25. Measurements and MoniPoli rules associated to AAA metric ALC

The AAA metrics defined above implement NIST and CCM security controls presented in the following table.

Control Family/Group	Control Name	Control ID	Security metric		
			SDA	ARGF	ALC
NIST					
Access Control	ACCESS CONTROL POLICY AND PROCEDURES	AC-1	✓		
	ACCOUNT MANAGEMENT | DYNAMIC PRIVILEGE MANAGEMENT	AC-2(6)	✓		
	ACCOUNT MANAGEMENT | ROLE-BASED SCHEMES	AC-2(7)	✓		
	ACCOUNT MANAGEMENT | DYNAMIC ACCOUNT CREATION	AC-2(8)	✓		
	ACCOUNT MANAGEMENT | ACCOUNT MONITORING / ATYPICAL USAGE	AC-2(12)	✓		
	ACCESS ENFORCEMENT	AC-3	✓		
	ACCESS ENFORCEMENT | ROLE-BASED ACCESS CONTROL	AC-3(7)	✓		
	UNSUCCESSFUL LOGON ATTEMPTS	AC-7	✓		
	AUDIT AND ACCOUNTABILITY POLICY AND PROCEDURES	AU-1	✓		
	AUDIT EVENTS	AU-2	✓		
	AUDIT EVENTS | REVIEWS AND UPDATES	AU-2(3)		✓	
	CONTENT OF AUDIT RECORDS	AU-3			✓
CCM					
Identity & Access Management	Credential Lifecycle / Provision Management	IAM-02	✓		
	Policies and Procedures	IAM-04	✓	✓	✓

Table 26. Mapping of AAA metrics to NIST and CCM security controls

4.5.1.3. Remediation

As introduced in deliverable D4.3.2 and summarized in Section 3.2, each measurement related to a security metric defines one monitoring event. The following table lists all possible monitoring events related to AAA metrics that can be detected by the Monitoring module.

ID	Condition	Affected metrics	Event type
AAA-E1	oauth_server_availability = no	SDA	violation
AAA-E2	ldap_server_availability = no	SDA	
AAA-E3	access_report_age > ARGF_value	ARGF	
AAA-E4	log_level != ALC_value	ALC	

Table 27. Monitoring events related to AAA metrics

Table 28 presents actions needed to remediate AAA alerts and violations.

ID	Description
AAA-A1	Restart OAuth Server.
AAA-A2	Restart LDAP Server.
AAA-A3	Force generation of access report and check if the report is available.

Table 28. AAA remediation actions

The following table presents remediation plan for managing alerts and violations of AAA metrics. For details on the structure of a remediation plan and the remediation process see deliverable D4.3.2 or Section 3.2.

Event	**AAA-E1 (V)**		**AAA-E2 (V)**		**AAA-E3/AAA-E4 (V)**		
Step 1	AAA-A1		AAA-A2		AAA-A3		
	yes	no	yes	no	yes	no	
Step 2	O	N	O	N	O	AAA-A1	
						yes	no
Step 3						O	N

Table 29. Remediation plan for alerts and violations related to AAA metrics

4.5.1.4. Development

The development of the AAA mechanism started in Task 4.2, whose result was a preliminary architecture of the AAA package, illustrated in deliverable D4.2.2. Although the associated requirements were already covered, we updated the design to offer a more flexible solution based on the popular technologies currently adopted. Moreover, in this deliverable we addressed the remediation aspects, not analysed before.

4.5.2. Repository

As said, the AAA mechanism is implemented by the AAA package, available on Bitbucket at [21]. Moreover, the repository contains also an example of OAuth client at the link [22].

4.5.3. Installation

In this section, we illustrate how to install the AAA mechanism both by manually configuring all needed software components and by using Chef recipes.

4.5.3.1. Manual Installation

The manual installation requires the following prerequisites:

- Git client
- Maven

- Java 7
- Java web container (e.g., Apache Tomcat [42])
- OpenLDAP Server [43]
- MySQL [44]

In order to install the AAA package, the following steps must be accomplished:

- Clone the git repository at [21].
- Convert it into a Maven project.
- Execute the 'maven install' command in order to execute tests and to generate the artifact.

In case of usage of the Eclipse IDE, these steps are detailed as:

- Import project from git as an "existing eclipse project".
- Right-click on the project, click on "Run as", then click on "Maven install".

The *specs-mechanism-enforcement-aaa* project generates a "war" file. By executing the maven goal "install" on it, the artifact is stored automatically in the "/target" folder. In order to use this component, the "war" file must be installed in a web container. If Apache Tomcat is used, the "war" file must be copied into the "webapps" folder that is inside the installation directory of Apache Tomcat.

As said, the AAA mechanism relies upon an LDAP directory service for storing user information. Therefore, an OpenLDAP server must be configured. To do this, the following steps must be carried out:

- Change permission of server configuration file:
  - chmod 777 usr/local/etc/openldap/slapd.conf
- Include schemas into LDAP Server:
  - include /usr/local/etc/openldap/schema/cosine.schema
  - include /usr/local/etc/openldap/schema/inetorgperson.schema
  - include /usr/local/etc/openldap/schema/java.schema
- Enable debug:
  - loglevel -1
- Define the database:
  - db ldif
  - suffix "dc=specs,dc=eu"
  - rootdn "cn=Manager,dc=specs,dc=eu"
  - rootpw pass
  - directory /usr/local/var/openldap-data
- Add the DIT (Directory Information Tree):
  - Move the folder "dc=specs,dc=eu" under the path /usr/local/var/openldap-data
- Launch the OpenLDAP server:
  - /usr/local/libexec/slapd -d 1 -f usr/local/etc/openldap/slapd.conf -h ldap://localhost:389

After configuring OpenLDAP, it is necessary to install and configure the database used by the DB App component. The database can be created by running the script *schema.sql*:

- mysql -u username -p database_name < schema.sql

Finally, the XACML policy called "PolicySpecsApp.xml" has to be moved under the folder /opt/xacml_policy.

4.5.3.2. Chef Recipe Installation

In addition to illustrating the steps to manually install the AAA mechanisms, we also provide in this section a brief guide to install the mechanisms by means of a Chef recipe (available at [5]).

This process requires to have a Chef Server installed and properly configured, and a Chef Workstation from which it is possible to execute the bootstrap of each target node.

The command to run on the workstation is the following:

```
knife bootstrap <public_ip_address_of_the_node_that_will_hosts_aaa> -x <chef_user_name> -P <chef_user_password> --node-name <node_name> --run-list 'recipe['AAA:server']'
```

4.5.4. Usage

As discussed in the overview section (Section 4.5.1), an OAuth Client must be included in the client application in order to communicate with the OAuth Server. In the following subsections, we illustrate all the operations that must be carried out by the OAuth Client to enable the flow described in Section 4.5.1, including the initial registration of the user and of the client, the authentication, and authorization, the submission of an access request and the support for new resources.

4.5.4.1. OAuth Client and User registration

In order to register a new OAuth Client on the OAuth Server, it is necessary to access the web interface provided by the OAuth Server at */specs-oauth2-server/* with the role of "Admin" and open the registration page under the path: */specs-oauth2-server/regapp*. The client information will be stored in the database managed by the App DB component and the client will be assigned a *clientId* and a *clientSecret*.

User registration can be done on the same URL. The user profile information will be stored in the LDAP server and the user will be assigned a set of credentials (a username and a password).

4.5.4.2. Authentication and Authorization

In order to obtain authentication and authorization for an access request, the OAuth Client has to call the *Authorization* end-point (/specs-oauth2-server/oauth/authorize) of the OAuth Server. The parameters to send into the GET request are:

1. client_id
2. redirect_uri
3. response_type
4. state

After the login phase (authentication), an authorization page is displayed to the user, who must give consent to access the protected resources. After the authorization, an authorization code is sent to the redirect_uri.

At this point, the OAuth client invokes the *Token* end-point (/specs-oauth2-server/oauth/token) to exchange the authorization code and receive the authorization token. To do this, a POST request at the token end-point is needed, with the parameters:

1. Authorization Header (contains the clientId and clientSecret Base64 encoded)
2. grant_type
3. redirect_uri
4. code

If the code exchange ends successfully, the redirect_uri is called and the OAuth token is sent by the OAuth Server to the client.

4.5.4.3. Access to Protected Resource

Once the OAuth token has been received, the Client Application can access the user profile or validate the token to verify the successful user authentication.

In particular, to validate the token, the *Validate_Token* end-point (/specs-oauth2-server/oauth/*check_token*) has to be called with the OAuth token as parameter.

To access the user profile (protected resource), the *User Profile* end-point (*/specs-oauth2-server/me)* must be called, with the OAuth token as Authorization Header.

4.5.4.4. Add a new Protected Resource

As said, the Resource Server is configured to manage a limited set of resources, namely the user profile information, with two built-in roles (admin and user). In order to extend the management to more resources and roles, the AAA mechanisms use the Authorization Service, which implements an XACML-based authorization.

As said before, during the mechanisms' installation, the default XACML policy file (PolicySpecsApp.xml) has to be moved under the folder /opt/xacml_policy. This file can be updated to update the authorization policy evaluated by the PDP.

For example, if the resource "home_reserved.jsp" needs to be supported, the following lines must be added to the file:

```
...
<Resource>
      <ResourceMatch      MatchId="urn:oasis:names:tc:xacml:1.0:function:regexp-
string-match">
            <AttributeValue   DataType="http://www.w3.org/2001/XMLSchema#string">
http://localhost:8080/specs-application-test/home_reserved.jsp </AttributeValue>
            <ResourceAttributeDesignator
DataType=http://www.w3.org/2001/XMLSchema#string
AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"/>
      </ResourceMatch>
</Resource>
...
```

5. Conclusions

Automatically enforcing SLAs in the cloud is a challenging task. The SLA enforcement tools have to (i) determine which features that are required from an End-user are implementable (if at all) with which CSPs, and (ii) how to automatically acquire cloud resources and deploy and configure additional mechanisms (if needed) that fulfil commitments in the signed SLA. Furthermore, the SLA enforcement tools have to support reconfigurations that (i) might be needed due to the changes in the SLA required either by the End-user (if the End-user wants to renegotiate her/his SLA) or by the CSP (due to the changes in the provided service), or (ii) are needed due to an unsuccessful remediation of an SLA violation. Although the cloud community is working on specifying security parameters in SLAs (see [24], [25], [26], [27]), and despite the fact that at the state of the art there were/are researchers actively working on enforcing security and adopting SLAs in the cloud [28], there is still a lack of tools that would enable their automated enforcement and monitoring (to the best of our knowledge, there are no solutions available that would automatically enhance the security level of cloud services through security SLAs). To this end, we have advanced the current state-of-the-art by developing a security-driven planning process adopted to determine the optimum deployment of security related software components [6]. Our innovative solution is able to automatically acquire and configure cloud resources to (optimally) deploy security-related software components for the enforcement of the security SLOs included in a signed SLA. The proposed approach founds on (i) matching customers' security requirements reported in the SLA with a set of security mechanisms offered as a service (Security-as-a-Service) and on (ii) automatically generating and implementing an allocation plan for the actual deployment of software components providing the desired security mechanisms. We have developed a set of components in SPECS that offer these exact functionalities. The Enforcement Planning and Implementation components first take as an input a set of End-user's security requirements (elicited during the SLA negotiation phase) and verify their feasibility according to CSP's capabilities and offers. Then, the Enforcement components prepare configuration details (the so-called implementation plan), and finally acquire cloud resources and deploy services according to the implementation plan in an automated way.

The automated enforcement of SLAs in the cloud only partially serves the needs of End-users and CSPs. There is a need for tools and mechanisms that are able to automatically react to eventualities that may threaten the fulfilment of the SLA commitments (e.g. cyber-attacks, system failures, changes in regulations). The detection, analysis, and reaction to potentially harmful security related events are thus essential activities to be carried out by CSPs in order to provide End-users with a trustworthy service and also fulfil the agreed upon assurance levels. In the Quality of Service context, SLAs usually include parameters (e.g., availability) that can be the subject of a progressive degradation of the acquired cloud service that imply an SLA violation (e.g., a system failure may anticipate degradation of the performance indicators and can thus result in a violation of the committed availability). Moreover, the CSPs are able to monitor such parameters and are able to automatically remediate incidents/failures that affect them (e.g., set up new resources). However, in the security context, the detection of such events (and even more so their automated remediation) is mostly unexplored. This is due to the fact that security parameters usually lack the aforementioned progressive degradation of the service (for example, if an attacker breaks an encryption key, there is no apparent indicator that would announce a security breach and the protected information is exposed). There are some works available, that tackle the problem of

the automated SLA remediation (for example, [29], [30], [31], [32]), but none of them is focused on security aspects. To this end, SPECS introduces an innovative approach to automatically react to events that might or do entail a violation of security commitments specified in the security SLA [35]. The designed methodology advances the current state-of-the-art by analysing detected events and selecting remediation actions to be executed in order to avoid or recover from an invalidation of any clause of the SLA. We have developed a set of components that enables the proposed automated SLA remediation. The Enforcement Diagnosis component can analyse any detected event that may present a potential or an actual SLA violation, and the RDS component can determine an optimal remediation plan to prevent or recover from it. The Implementation component executes the remediation plan in an automated fashion, similarly as in the SLA enforcement phase.

In particular, in the last six months of the project, we have added functionalities related to implementing renegotiated SLAs to the Planning and the Implementation component. We have been working on improving the quality of the code in terms of security, scalability, and mostly performance, and addressing issues that were arising from the integration activities. Additionally, we have improved three security mechanisms already presented in deliverable D4.3.2 (namely the E2EE, DBB, and SVA), and developed two new security mechanisms (namely AAA and DoS).

In the table below we present the list of objectives associated to the task T4.3 and report the outcomes which verify the benefits of the results achieved in this task in the entire duration of the project.

Objective	**Result**
SO4.1: Design and implement services to **check the effective availability of security features** and provide this information to the Negotiation mechanism, determining the possibility of the SLA fulfilment with respect to security	• We have improved the current state of the art by designing an innovative (security) **SLA-based approach to the generation of valid supply chains** (i.e., automated verification of implementability of an SLA) [6]. • The developed **Planning component** [8] enforces the verification of effective availability of security features thus supporting the SLA negotiation phase by implementing the designed process of generating valid supply chains. • We have designed the **mechanism data model** [33] that supports the process of verifying feasibility of an SLA.
SO4.2: Design and implementation of Cloud services able to check the running software stack and **activate the opportune actions in order to respect an agreed cloud SLA**	• We have designed the **SLA-based approach to generating implementation plans** to enable automated implementation of SLAs (presented in deliverable D4.3.2). • The designed **implementation plan data model** [34] and the **mechanism data model** [33] support the process of automated SLA implementation. • The developed **Planning component** [8] enforces the designed process of generating implementation plans thus supporting the SLA implementation phase. • We have designed the **automated SLA implementation process** (i.e., the process of executing implementation plans) in terms of automatically acquiring cloud resources and automatically managing deployment of mechanisms on top of them and their

	configuration (elaborated in deliverable D4.3.2). • The developed **Implementation component** [9] (integrated with the **Broker component** [10]) enforces the activation of opportune actions in order to respect an agreed cloud SLA by implementing the designed SLA implementation process.
SO4.3: Provide a sustained QoSec during the life cycle of the application/service, as agreed on the Cloud SLA (cf., negotiation stage)	• We have designed the **automated SLA remediation process** that comprises the analysis of detected security incidents/system failures and identification and execution of associated remediation plans (elaborated in deliverable D4.3.2 and presented in [35]. • The designed **remediation plan data model** [36] and the **mechanism data model** [33] support the process of automated SLA remediation phase. • The developed **Diagnosis component** [8], the **RDS component** [12], and the **Implementation component** [9] (integrated with the **Broker component** [10]) provide a sustained QoSec during the SLA life cycle by implementing the designed SLA remediation phase.
SO4.4: Offer additional security services to end users in order to sustain a minimum required QoSec	• We have **developed (or adjusted and integrated) a set of negotiable security mechanisms** that offer additional security services to end users: o **WebPool** [13], [14] (see deliverable D4.3.1 for initial and D4.3.2 for final prototype) o **TLS** [15] (see D4.3.2) o **SVA** [16], [17], [18], [19], [20] (see deliverable D4.3.1 for initial, D4.3.2 for intermediate, and Section 4.3 for final prototype) o **AAA** [21], [22] (see Section 4.5 for the prototype and deliverable D5.4 for the AAA-as-a-Service application) o **DoS** [23] (see Section 4.4 for the prototype) o **DBB & E2EE** [1], [2], [3], [4] (see deliverables D4.3.2 for initial prototype, D5.2.1 for improved functionalities, D5.2.2 for final prototype, and Section 4.2 for elaboration on how final prototypes are automatically managed during the SLA life cycle) • The designed **mechanism data model** [33] enables automated management of developed/integrated mechanisms. • The developed security mechanisms are part of the SPECS solution portfolio [37].

Table 30. Objectives and results of the task T4.3

The developed approach to verifying feasibility of an SLA (presented in deliverable D4.3.2 and in [6]) currently deals with SLAs sequentially, i.e. separately for each End-user. Since dealing concurrently with the service request from various End-users could allow for a better resource planning, in future (as part of our exploitation activities) we intend to improve the algorithm to better exploit the multitenancy features.

The crucial part of the SLA remediation process (in security or any other domain) is a thorough analysis of the detected event that causes or anticipates an SLA violation. In SPECS,

the developed diagnosis process relies only on the information provided by the monitoring adapters deployed on the acquired cloud resources for each End-user/SLA separately. A more meaningful analysis of SLA violations and a better detection of SLA alerts (i.e., potential SLA violations) would be possible if some additional information would be considered as well (e.g., historic data, monitoring data gathered on other parts of the infrastructure acquired for other End-users/SLAs). In future (as part of our exploitation activities), our goal is to develop an improved version of the process of analysing detected SLA alerts and violations.

SPECS currently deals with provisioning of cloud services only with one cloud provider per End-user/SLA. Moreover, the goal of SPECS is to develop a framework for an automated management of the SLA life cycle without taking into account the business side of the service provisioning. SPECS offers a complete platform for the SLA management and for the enhancement of the security level of the provisioned cloud service, thus after the end of the project a good research direction would be to work towards enforcing SLAs in the multi-cloud environment taking into account even the costs associated to the provisioned services and the trade-offs among the cost, the performance, and the security of the provisioned service. Note that (i) different CSPs implement different security controls and (ii) not all security mechanisms, which can enhance the security level of the services offered by CSPs, can be implemented on every cloud resource. Therefore, supporting enforcement of SLAs with more than one CSP at the same time, additionally associating provisioning of such services not only with some sort of a security level score (as currently done for each CSP by the Security Reasoner component of the SLA Platform – see deliverable D2.3.2) but also a concrete service cost, and analysing cost-performance-security trade-offs, would give the End-user an opportunity to better identify an optimal service for her/his security requirements.

6. Bibliography

[1] SPECS, "*SPECS Enforcement E2EE Server*", 2015. Available online, https://bitbucket.org/specs-team/specs-mechanism-enforcement-e2ee-server, last accessed in April 2016.

[2] SPECS, "*SPECS Enforcement E2EE Client*", 2015. Available online, https://bitbucket.org/specs-team/specs-mechanism-enforcement-e2ee-client, last accessed in April 2016.

[3] SPECS, "*SPECS Enforcement E2EE Auditor*", 2015. Available online, https://bitbucket.org/specs-team/specs-mechanism-monitoring-e2ee-auditor, last accessed in April 2016.

[4] SPECS, "*SPECS Enforcement E2EE Monitoring Adapter*", 2015. Available online, https://bitbucket.org/specs-team/specs-mechanism-monitoring-e2ee-adapter, last accessed in April 2016.

[5] SPECS, "*SPECS Core Enforcement Repository"*, 2015. Available online, https://bitbucket.org/specs-team/specs-core-enforcement-repository/overview, last accessed in April 2016.

[6] V. Casola, A. De Benedictis, M. Erascu, J. Modic, M. Rak, "Automatically Enforcing Scurity SLAs in the Cloud", 2016. IEEE Transactions on Services Computing – Special Issue on Security and Dependability of Cloud Systems and Services, 2016. Available online, http://doi.ieeecomputersociety.org/10.1109/TSC.2016.2540630, last accessed in June 2016.

[7] Chef Software Inc., "*Chef*", 2008. Available online, https://www.chef.io/, last accessed in March 2016.

[8] SPECS, "*SPECS Core Enforcement Planning*", 2015. Available online, https://bitbucket.org/specs-team/specs-core-enforcement-planning, last accessed in April 2016.

[9] SPECS, "*SPECS Core Enforcement Implementation*", 2015. Available online, https://bitbucket.org/specs-team/specs-core-enforcement-implementation, last accessed in April 2016.

[10] SPECS, "*SPECS Core Enforcement Broker*", 2015. Available online, https://bitbucket.org/specs-team/specs-core-enforcement-broker, last accessed in April 2016.

[11] SPECS, "*SPECS Core Enforcement Diagnosis*", 2015. Available online, https://bitbucket.org/specs-team/specs-core-enforcement-diagnosis, last accessed in April 2016.

[12] SPECS, "*SPECS Core Enforcement RDS*", 2015. Available online, https://bitbucket.org/specs-team/specs-core-enforcement-rds, last accessed in April 2016.

[13] SPECS, "*SPECS Mechanism Enforcement WebPool*", 2015. Available online, https://bitbucket.org/specs-team/specs-mechanism-enforcement-webpool, last accessed in April 2016.

[14] SPECS, "*SPECS Mechanism Monitoring WebPool Adapter*", 2015. Available online, https://bitbucket.org/specs-team/specs-mechanism-monitoring-webpool-adapter, last accessed in April 2016.

[15] SPECS, "*SPECS Core Enforcement TLS*", 2015. Available online, https://bitbucket.org/specs-team/specs-core-enforcement-tls/, last accessed in April 2016.

[16] SPECS, "*SPECS Mechanism Enforcement SVA Core*", 2015. Available online, https://bitbucket.org/specs-team/specs-mechanism-enforcement-sva core, last accessed in April 2016.

[17] SPECS, "*SPECS Mechanism Enforcement SVA Dashboard*", 2015. Available online, https://bitbucket.org/specs-team/specs-mechanism-enforcement-sva dashboard, last accessed in April 2016.

[18] SPECS, "*SPECS Mechanism Monitoring SVA*", 2015. Available online, https://bitbucket.org/specs-team/specs-mechanism-monitoring-sva, last accessed in April 2016.

[19] SPECS, "*SPECS Mechanism Enforcement SVA Vulnerability Manager*", 2015. Available online, https://bitbucket.org/specs-team/specs-mechanism-enforcement-sva vulnerability manager, last accessed in April 2016.

[20] SPECS, "*SPECS Mechanism Monitoring OpenVAS*", 2015. Available online, https://bitbucket.org/specs-team/specs-mechanism-monitoring-openvas, last accessed in April 2016.

[21] SPECS, "*SPECS Mechanism Enforcement AAA*", 2015. Available online, https://bitbucket.org/specs-team/specs-mechanism-enforcement-aaa, last accessed in April 2016.

[22] SPECS, "*SPECS Mechanism Enforcement AAA Client*", 2015. Available online, https://bitbucket.org/specs-team/specs-mechanism-enforcement-aaa-client, last accessed in April 2016.

[23] SPECS, "*SPECS Mechanism Enforcement DoS Detection and Mitigation*", 2015. Available online, https://bitbucket.org/specs-team/specs-mechanism-enforcement-dos, last accessed in April 2016.

[24] European Network and Information Security Agency, "*Survey and Analysis of Security Parameters in Cloud SLAs across the European Public Sector*", 2011. Available online, https://www.enisa.europa.eu/activities/Resilience-and-CIIP/cloud-computing/survey-and-analysis-of-security-parameters-in-cloud-slas-across-the-european-public-sector, last accessed in March 2016.

[25] International Organization for Standardization, "*Information Technology – Cloud Computing – Service Level Agreement (SLA) Framework and Terminology (Draft), ISO/IEC 19086*", 2014.

[26] National Institute of Standards and Technology (NIST), "*Security and Privacy Controls for federal Information Systems and Organizations, NIST 800-53v4*", 2014. Available online, http://csrc.nist.gov/publications/drafts/800-53-rev4/sp800-53-rev4-ipd.pdf, last accessed in March 2016.

[27] European Commission, "*Cloud Service Level Agrement Standardization Guidelines, C-SIG SLA 2014*", 2014. Available online, https://ec.europa.eu/digital-single-market/en/news/cloud-service-level-agreement-standardisation-guidelines, last accessed in March 2016.

[28] DPSP Cluster, "*Data Protection, Secrity and privacy in Cloud*", 2016. Available online, https://eucloudclusters.wordpress.com/data-protection-security-and-privacy-in-the-cloud/, last accessed in March 2016.

[29] I. Brandic, V. C. Emeakaroha, M. Maurer, S. Dustdar, S. Acs, A. Kertesz, G. Kecskemeti, "*LAYSI: A Layered Approach for SLA-Violation Propagation in Self-Manageable Cloud Infrastructures*", 2010. In proceedings of the COMPSACW'10, the 34th Annual IEEE Computer Software and Applications Conference Workshops, pp. 365–370. Available online, http://www.infosys.tuwien.ac.at/staff/sd/papers/Compsac%202010%20I.%20Brandic.pdf, last accessed in March 2016.

[30] O. F. Rana, M. WarnierT. B. Quillinan, F. Brazier, D. Cojocarasu, "*Managing violations in service level agreements*", 2008. In Grid Middleware and Services, Springer, US, pp. 349 – 358.

[31] Z. Zhang, L. Liao, H. liu, G. Li, "*Policy-Based Adaptive Service Level Agreement Management for Cloud Services*", 2014. In proceedings of the ICSESS'14, the 5th IEEE International Conference on Software Engineering and Service Science, pp. 496–499.

[32] SLA@SOI Project (FP7-216556, Empowering the Service Economy with SLA-aware Infrastructures), 2011. Available online, http://sla-at-soi.eu/, last accessed in March 2016.

[33] SPECS, "*Mechanism Data Model*", 2015. Available online, http://www.specs-project.eu/resources/schemas/json/mechanism.json, last accessed in March 2016.

[34] SPECS, "*Implementation Plan Data Model*", 2015. Available online, http://www.specs-project.eu/resources/schemas/json/plan.json, last accessed in March 2016.

[35] R. Trapero, J. Modic, M. Stopar, A. Taha, N. Suri, "*A Novel Approach to Manage Cloud Security SLA Incidents*". Submitted to the Future Generation Computer Systems – Special Issue on Cloud Incident Management and Disaster Recovery, 2016. Available online, http://dx.doi.org/10.1016/j.future.2016.06.004, last accessed in June 2016.

[36] SPECS, "*Remediation Plan Data Model*", 2015. Available online, http://www.specs-project.eu/resources/schemas/json/remplan.json, last accessed in March 2016.

[37] SPECS, "*SPECS Solutions Portfolio*", 2015. Available online, http://www.specs-project.eu/solutions-portofolio/, last accessed in March 2016.

[38] SPECS, "*Reconfiguration Data Model*", 2015. Available online, http://www.specs-project.eu/resources/schemas/json/reconfiguration.json, last accessed in March 2016.

[39] Internet Engineering Task Force (IETF), D. Hardt, "*The OAuth 2.0 Authorization Framework*", 2012. Internet Request for Comments, RFC 6749. Available online, http://tools.ietf.org/html/rfc6749, last accessed in March 2016.

[40] Internet Engineering Task Force (IETF), K. Zeilenga, "*Lightweight Directory Access*

Protocol", 2006. Internet Request for Comments, RFC 4510. Available online, http://tools.ietf.org/html/rfc4510, last accessed in March 2016.

[41] OASIS, "*OASIS eXtensible Access Control Markup Language (XACML) TC*", 2011. Available online, https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml, last accessed in March 2016.

[42] The Apache Software Foundation, "*Apache Tomcat*", 2016. Available online, http://tomcat.apache.org/, last accessed in March 2016.

[43] OpenLDAP Foundation, "*OpenLDAP*", 2014. Available online, http://www.openldap.org/, last accessed in March 2016.

[44] Oracle, "*Download MySQL Community Server*", 2016. Available online, https://dev.mysql.com/downloads/mysql/, last accessed in March 2016.

[45] SPECS, "*SPECS Mechanism Monitoring DoS Detection and Mitigation*", 2015. Available online, https://bitbucket.org/specs-team/specs-mechanism-monitoring-dos, last accessed in April 2016.

[46] OSSEC Project Team, "*OSSEC*", 2016. Available online, http://ossec.github.io/, last accessed in April 2016.

[47] OWASP Project, "*OWASP Zed Attack Proxy Project*", 2016. Available online, https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project, last accessed in April 2016.

[48] SPECS, "*SPECS Mechanism Monitoring OSSEC*", 2015. Available online, https://bitbucket.org/specs-team/specs-mechanism-monitoring-ossec, last accessed in April 2016.

Appendix 1. Developing and adding a security mechanism

This guide aims at illustrating how a developer can develop a new security mechanism and integrate it into the SPECS framework. Security mechanisms can be developed using already existing commercial of-the-shelf components and/or from scratch: this guide outlines how to do both; we present (i) how each mechanisms should be prepared (either how to adjust existing solution or how to develop a new one) and (ii) how it should be integrated.

The development and integration process is depicted in Figure 26. First (*Step 1*), the developer has to define the cloud service to be security hardened with the developed mechanism (e.g., secure storage, secure web container) and then it has to define specific security features that the mechanism can enforce and/or monitor (e.g., enforce redundancy, monitor availability of servers). Afterwards (*Step 2*), the developer has to define the mechanism's architecture. As we will discuss later, each mechanism should comprise at least one monitoring component that continuously evaluates the parameters associated to the features that the mechanism enforces/monitors. The developer has to specify all details that are associated to the process of automated deployment of the mechanism (*Step 3*) and all details that are associated to the process of automated remediation of alerts and violations of commitments related to the security features enforced/monitored by the developed mechanism (*Step 4*). Since the automated management in SPECS is orchestrated with Chef, the developer has to create recipes for all enforcement and remediation actions (and organize them in a Chef cookbook), and register them in the SPECS Chef repository (*Step 5*). Finally, the developer has to provide the above defined mechanism's properties in a metadata file and register it in the SLA Platform (in the Service Manager component; see deliverable D1.4.1). Further details are elaborated in the following subsections.

Figure 26. SPECS security mechanism development and integration process

Step 1: Define offered services

In this step, the developer has to define the service that can be security hardened with the developed mechanism. As depicted in Figure 27, this comprises the definition of the following attributes:

- **Service:** First, the developer has to identify the cloud service for which the developed mechanism provides security enhancements. In SPECS there are currently two cloud services that can be acquired and security hardened with SPECS security mechanisms, namely the secure web container (for acquisition of virtual machines and deployment of web servers) and the secure storage service (for acquisition of cloud storage).
- **Capabilities**: Each mechanism has a specific set of functionalities. For example, some mechanism may enforce client-side encryption, some mechanism may enforce redundancy of web servers, some mechanism may offer software vulnerability assessment of cloud resources, and some mechanism may monitor configurations of the TLS protocol. In SPECS, these functionalities are grouped into different capabilities. The developer has to define a capability that is enforced and/or monitored with the

developed mechanism. In the SLA negotiation phase, capabilities are the first attributes to be chosen by the End-user for the preferred cloud service.

- **Security controls**: The developer has to specify how the developed security mechanism implements the defined capability. To this end, the developer has to identify the set of security controls that the mechanism can implement. In SPECS, we use two different security control frameworks, namely the NIST's SP 800-53v4[2] and the CSA's CCMv3[3]. In the SLA negotiation phase, the End-user selects the preferred security controls for the chosen capabilities.
- **Security metrics**: Each mechanism can enforce and monitor security related features (e.g., provides and monitors client-side encryption to guarantee data confidentiality) or it can only monitor them (e.g., monitors checksums to guarantee data integrity). In any case, each mechanism's enforcement or monitoring functionality can be offered to the End-user through security metrics. The developer has to identify existing metrics in the SPECS Metric Catalogue application (which stores information about all metrics used in SPECS) or define the set of new security metrics that the mechanism can enforce/monitor (with all associated details like units, possible values, operators, etc.) and map them to the identified security controls. In the SLA negotiation phase, the End-user specifies SLOs, which means that she/he selects the set of preferred security metrics for the chosen security capabilities and sets their preferred thresholds.
- **Measurements**: In order to enable continuous evaluation of negotiated security features (to enable continuous verification that the negotiated thresholds for the negotiated security metrics are respected), the developer has to specify a set of measurements for each security metric. In other words, the developer has to specify how each security metric is to be evaluated. We have two types of measurements, namely *basic* and *additional* ones. Basic measurements are used to directly evaluate whether an SLO is respected or not, whereas the additional measurements for a metric enable detection of potential violations. The requirement is that each metric should have defined at least one basic measurement. In the SLA implementation phase, SPECS configures monitoring components to continuously evaluate measurements associated to the negotiated security metrics.

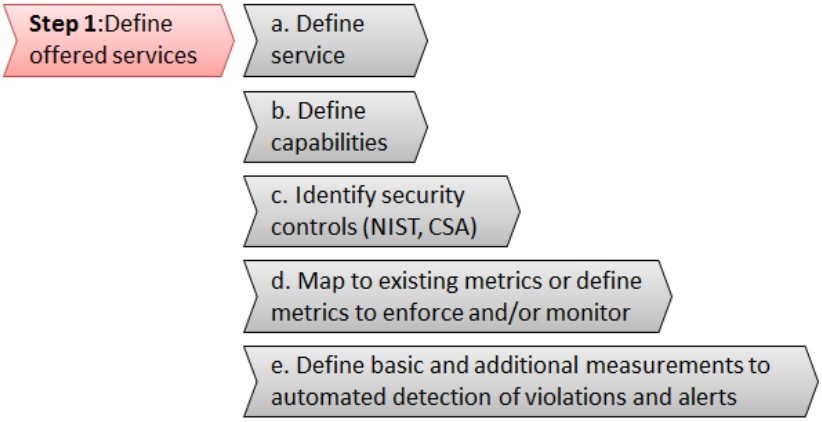


Figure 27. Defining offered services

[2] http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-53r4.pdf
[3] https://cloudsecurityalliance.org/group/cloud-controls-matrix/

Let us consider a simple example. Let us assume that the developer wants to develop and integrate a security mechanism named Software Vulnerability Assessment (SVA) mechanism that would offer software vulnerability scans. The mechanism would download a list of disclosed vulnerabilities from a public repository, generate a list of vulnerabilities, perform a vulnerability scan on a VM, and generate a scanning report.

First, the developer identifies the *Secure Web Container* service because the developed mechanism offers vulnerability scans of VMs. In the next step, the developer defines the *Software Vulnerability Assessment* capability and identifies three security controls that the mechanism implements: CA-7 (Security Assessment and Authorization – Continuous Monitoring) and RA-5 (Risk Assessment – Vulnerability Scanning) from the NIST, and control TVM-02 (Threat and Vulnerability Management – Vulnerability/Patch Management) from the CSA. Next, the developer defines a security metric for the developed mechanism, namely the *Scanning Frequency*, with which the End-user specifies how often SPECS should perform vulnerability scans on the acquired VMs. Finally, the developer defines a set of measurements with which SPECS can continuously evaluate fulfilment of commitments associated to this metric. In particular, the developer defines the basic measurement *Age of the Scanning Report*, which directly shows whether the negotiated scanning frequency is respected or not, and the additional measurements *Scanner Availability*, *Vulnerability List Availability*, and *Repository Availability*, which can indicate that a violation of the SLO, which defines the scanning frequency, might be violated (for example, if at one point the scanner installed on the VM for the purpose of executing vulnerability scans is unresponsive, or the list of published vulnerabilities is unavailable, or the repository from which the published vulnerabilities are extracted is unavailable, this may mean that at the scheduled time the vulnerability scan will not be executed – this may cause an SLA violation).

Note that the capability, controls, and metrics are crucial parts for the SLA negotiation phase, whereas the measurements are the base for the SLA implementation and SLA monitoring phase.

Step 2: Define architecture

In this step (depicted in Figure 28), which is crucial for the SLA implementation step, the developer has to define and develop all components that are needed to enforce and/or monitor the defined security metrics. In particular, each mechanism has to comprise a set of components with which guaranteed security assurances can be enforced (at least one component is needed to enforce the defined security metrics) and monitored (at least one component is needed to take measurements associated to the defined security metrics).

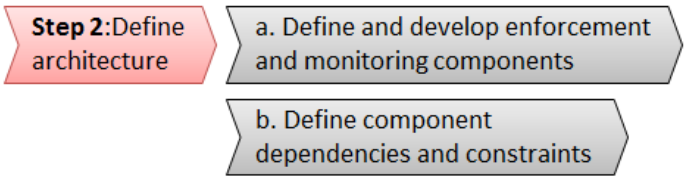


Figure 28. Defining the architecture

In order to automate SLA monitoring and SLA remediation phase, the monitoring components of the security mechanism have to be able to continuously collect the monitoring data and

report results in a specific format (the monitoring data should be reported in the format of the *eventhubresult.json* introduced in deliverable D1.3) to the Event Hub (component of the Monitoring module; see deliverable D3.4.1).

Apart from these requirements, there are no real limitations to what the architecture of the security mechanism should look like. When defining and developing a mechanism, the developer can reuse existing solutions or can develop all components from scratch.

In order to enable the SLA implementation phase, the developer has to consider and specify all dependencies among components (i.e., what are the dependencies among different components, which components need to be deployed together, which components are incompatible) and constraints associated to their deployment (e.g., which components need to be deployed for the enforcement/monitoring of each security metric, how many instances of each component are needed for the enforcement/monitoring of each security metric).

For the example considered in the previous step (for the SVA mechanism) the developer would have to define and develop the following components:

- SVA Scanner: This component is needed to execute vulnerability scans on a VM.
- SVA Enforcement: In order to perform vulnerability scans, the SVA Scanner needs a list of published vulnerabilities. They are maintained (continuously downloaded from a public repository of disclosed software vulnerabilities) by the SVA Enforcement component and fed to the SVA Scanner.
- SVA Monitoring: This component continuously evaluates the measurements specified for the security metrics of the SVA mechanism.
- SVA Dashboard: In order to present scanning reports to the End-user, the SVA mechanism comprises a dashboard component.

For the defined components, the developer would specify the following dependencies and constraints:

- One instance of the SVA Scanner, the SVA Enforcement, and the SVA Monitoring component should be installed on each VM hosting a web server.
- One instance of the SVA Dashboard component should be installed for each acquisition of the SVA mechanism.
- The SVA Dashboard component is incompatible with the rest of the SVA components and should thus be deployed on a separate VM.

Step 3: Define implementation details

To enable the automated SLA implementation, the developer should specify all configuration details (as shown in Figure 29) for each component defined and developed for the security mechanism (these details should later be specified in the mechanism metadata file in the format of the *mechanism.json* introduced in deliverable D1.3):

- Component type: Each component is either an enforcement component (i.e., enforces security metrics), a monitoring adapter (i.e., takes measurements and thus monitors security metrics), a dashboard (i.e., serves as an interface for the End-user that presents different reports related to the security services provided by the mechanism), etc. The developer has to specify the type for each defined and developed component.
- Implementation step: In some cases it is important to specify whether some component should be deployed before or after another one. The developer has to

define the order in which all mechanism's components should be deployed (whether they can all be deployed in parallel or whether some set of components needs to be deployed in a certain sequence).

- Pool sequence number/pool ID: Some components should be (for security reasons) physically separated. For example, for data storage, the main database should be physically separated from the backup. For this purpose, we introduce the concept of pools of VMs, on which all the components are deployed. If two components should be separated, they should have different pool numbers/IDs assigned to them. The developer has to specify the pool number/ID for each defined and developed component.
- Recipes/cookbook: In SPECS, all deployment is orchestrated by Chef[4] (see deliverable D4.2.2). So the developer should prepare a list of Chef recipes (organized in a Chef cookbook) that are needed for an automated deployment and management of the mechanism (recipes for the automated installation and configuration of each mechanism's component).
- VM requirement: For each component of the mechanism, the developer should report all hardware requirements (e.g., minimal RAM/CPU in terms of minimal instance type as defined by Amazon, firewall rules).
- Constraints: The developer should consider any requirements and constraints in terms of incompatibilities and dependencies among components. For example, the WebPool mechanism in SPECS offers web servers (see deliverables D4.2.2 and D4.3.2). And since there are two different types of web servers available (Apache[5] and Nginx[6]), it is a natural constraint that the two web servers should not be installed on the same VM. All constraints should be prepared in a formal way, as described in Appendix 1 of deliverable D4.3.2 and specified in the *mechanism.json* format introduced in deliverable D1.3.

For the SVA mechanism considered above, the developer would denote the SVA Scanner and the SVA Enforcement mechanism as enforcement components, the SVA Monitoring as the monitoring component, and the SVA Dashboard as the component of the type dashboard. Since there are no deployment constraints in terms of the sequence of installation, they can all be deployed in parallel (all components have implementation step set to 1). Moreover, there are no limitations in terms of physical separation of VMs hosting the SVA components, therefore the pool numbers for all components are set to 1. Since the SVA mechanism, which aims to security harden a pool of web servers, can only be deployed on VMs hosting the web servers (which are enforced by the WebPool mechanism; see deliverables D4.2.2 and D4.3.2 for further details), the Pool ID is set to *WebPool* (indicating that the SVA mechanism is to be deployed in the same pool of VMs acquired for the WebPool mechanism). As for the rest of the properties, the developer has to prepare the set of recipes with which each component is automatically installed and configured (e.g., *Install SVA Scanner*, *Install SVA Monitoring*), specify VM requirements for each SVA component, and formalize the constraints defined

[4] https://www.chef.io/
[5] https://httpd.apache.org/
[6] https://www.nginx.com/

above (dependencies among SVA Scanner, SVA Enforcement, SVA Monitoring, and incompatibilities with the SVA Dashboard).

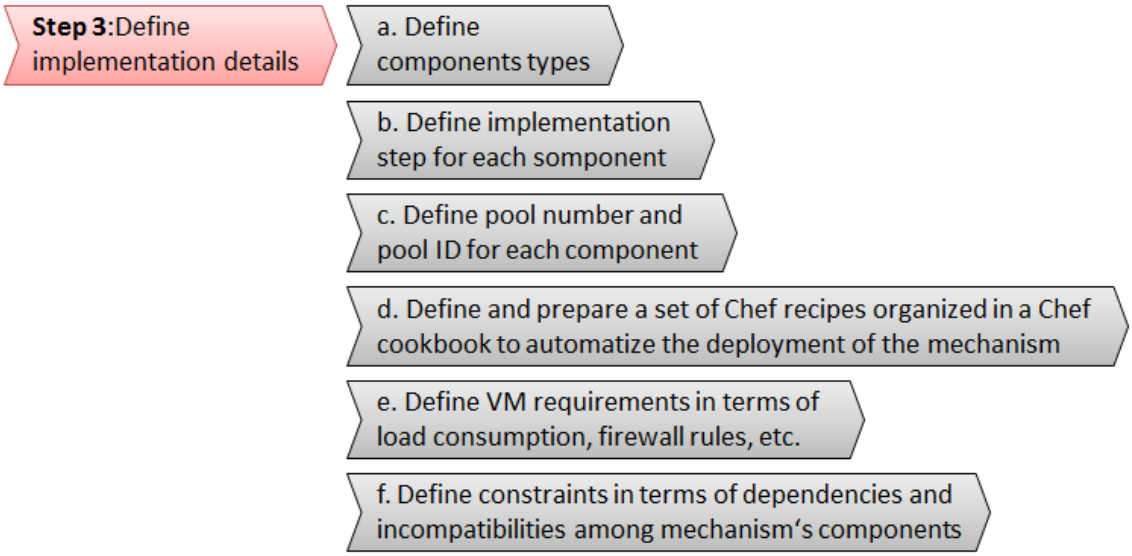


Figure 29. Defining implementation details

Step 4: Define remediation details

In this phase (illustrated in Figure 30), the developer has to define alerts and violations for each security metric that the mechanism can enforce and/or monitor. For each of the defined alerts and violations, the developer has to specify a list of actions with which the alert or violation can be automatically resolved. These actions have to be formalized and prepared in terms of Chef recipes.

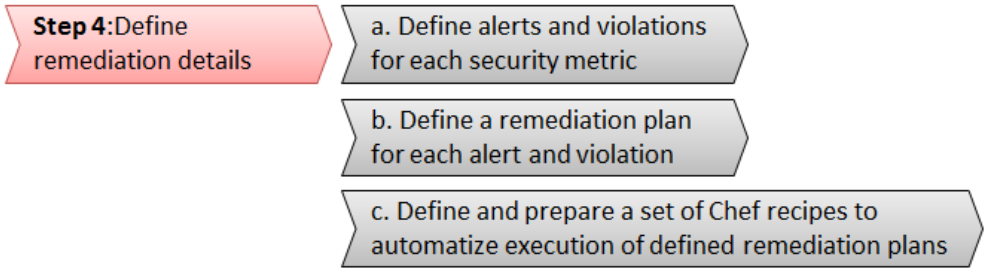


Figure 30. Defining remediation details

As mentioned in Step 1, each security metric is associated to one or more measurements. In the SLA implementation phase each metric and its value (negotiated by the End-user) are transformed into a set of measurements and corresponding thresholds (as defined by the mechanism's developer). For example, the metric *Scanning Frequency* defined in Step 1, is associated to one basic and three additional measurements as presented in Table 31.

The mechanism's monitoring components (in our case, the SVA Monitoring component) continuously evaluate the defined measurements and report their results to the SPECS Monitoring module. The Monitoring module (in particular, the Monitoring policy filter component; see deliverable D3.4.2) compares the measurement results with the thresholds specified during the SLA implementation phase. Whenever a measurement deviates from the

defined threshold, this occurrence indicates an SLA alert (if the deviated measurement is the additional one) or an SLA violation (if the deviated measurement is the basic measurement for the affected security metric). Thus for each measurement defined for the set of security metrics that the mechanism can enforce/monitor the developer should specify one monitoring event and determine its type (alert vs. violation).

Metric	*Scanning Frequency (SF)*	
SLO	*scanning_frequency = N hours*	
Measurement	Type	Threshold
scanning_report_age	basic	*scanning_report_age ≤ N*
scanner_availability	additonal	*repository_availability = yes*
vulnerability_list_availability		*vulnerability_list_availability = yes*
repository_availability		*repository_availability = yes*

Table 31. Measurements associated to the Scanning Frequency metric

For the *Scanning Frequency* security metric, the developer of the SVA mechanism has to define monitoring events for the three associated measurements presented in Table 31. If at any point the age of the scanning report is higher than the defined threshold, this represents an SLA violation. SLA alerts are raised whenever the SVA Monitoring component detects and the SPECS Monitoring module detects unavailability of the SVA Scanner, unavailability of the vulnerability list or unavailability of the vulnerability repository. The defined monitoring events are summarized in Table 32. Note that since each additional measurement can be used to evaluate the status of more than one security metrics, the developer has to map defined events to all affected security metrics.

Event ID	Event condition	Affected metrics	Event type
SVA-E1	*scanning_report_age > N*	Scanning Frequency	violation
SVA-E2	*repository_availability = no*		alert
SVA-E3	*vulnerability_list_availability = no*		
SVA-E4	*repository_availability = no*		

Table 32. Monitoring events associated to the measurements of the LUF metric

Each defined monitoring event should be accompanied by a specified remediation plan, i.e., a set of possible remediation actions and a clear sequence in which they should be executed. Note that each remediation action should be composed either of some monitoring action (i.e., to check a measurement) or one or more enforcement actions followed by a monitoring action (i.e., perform some reconfigurations and check a measurement related to them).

For example, in case of an alert associated to unavailability of the vulnerability list, the developer considers the procedure depicted in Figure 31. First, the SVA Monitoring component should verify whether the configured repository, from where the disclosed vulnerabilities are extracted, is available. In case the repository is responsive, the SVA Enforcement component should delete the old and generate a new list of vulnerabilities. If this action solves the issue, the alert is resolved; otherwise the End-user should be notified about the occurrence and warned that the SLA might be violated. If the initially configured repository for downloading published vulnerabilities is unavailable, the first step would be to reconfigure the repository and connect to an alternative one. If the new repository is

unavailable, the End-user is notified, otherwise the SVA Enforcement component tries to generate a new vulnerability list. If the generation is successful, the alert is resolved, otherwise the End-user is notified about the event and potential consequences.

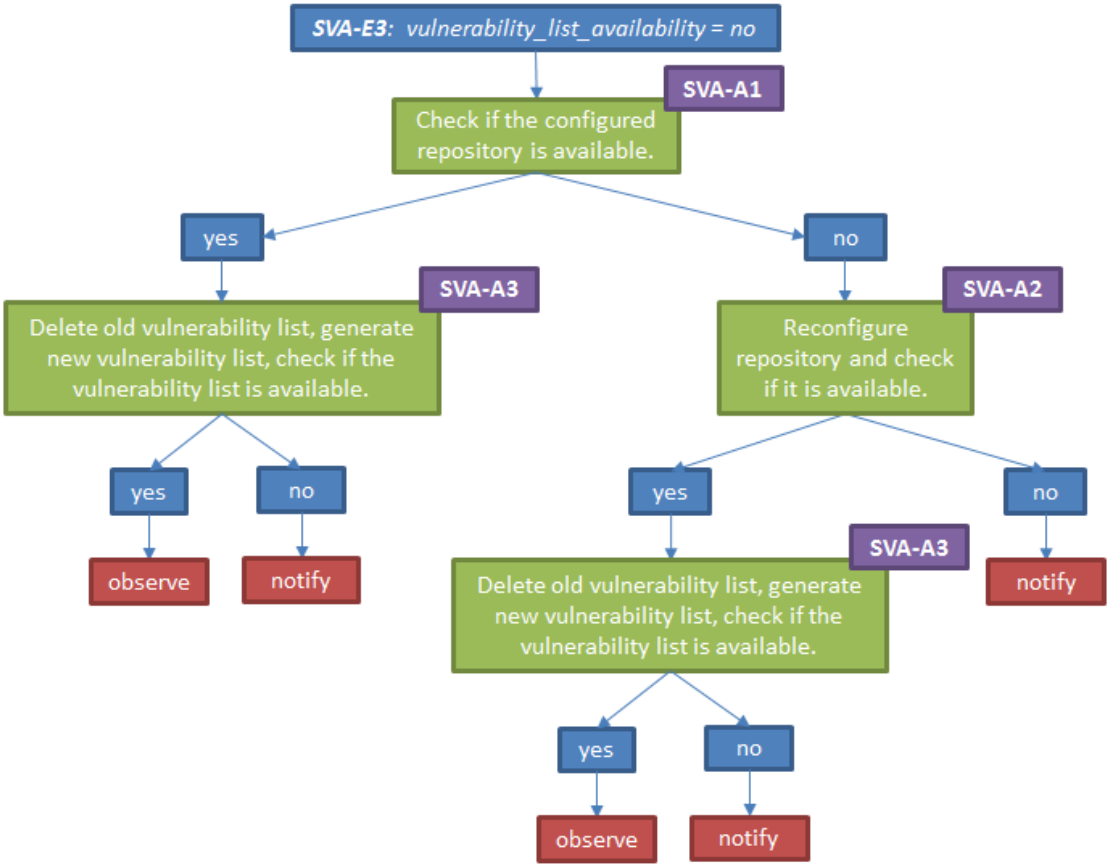


Figure 31. Remediation plan for one of SVA alerts

As seen in the diagram in Figure 31, a remediation plan comprises a set of remediation action (in green), which involve either a single monitoring action (like remediation action SVA-A1) or one or more enforcement actions followed by a monitoring action (like remediation actions SVA-A3 and SVA-A3). Note that the monitoring actions have to be defined in such a way, that there are always only two possible outcomes: whether the action indicates a partial (meaning that the performed reconfiguration is successful and that some further steps might resolve the issue) or full resolution (the alert/violation is resolved) of the issue or the action failed to at least partially resolve the alert/violation. Depending on the result of that measurement, SPECS decides on the next step.

Finally, to enable an automated execution of remediation plans, each remediation action has to be accompanied by a sequence of Chef recipes. More precisely, each enforcement and/or monitoring action involved in a remediation action has to be defined with a Chef recipe.

For the example above, the Table 33 lists the Chef recipes that the developer has to prepare for remediation of the alert SVA-E3.

ID	Description
SVA-R1	Check if the configured repository is available (i.e., invoke the SVA Monitoring component to take measurement *Repository Availability*).
SVA-R2	Delete old vulnerability list (i.e., trigger the SVA Enforcement component to delete the existing vulnerability list).
SVA-R3	Generate new vulnerability list (i.e., trigger the SVA Enforcement component to download published vulnerabilities from the configured repository and generate a new vulnerability list).
SVA-R4	Check if the vulnerability list is available (i.e., invoke the SVA Monitoring component to take measurement *Vulnerability List Availability*).
SVA-R5	Reconfigure repository (i.e., reconfigure the SVA Enforcement component to download published vulnerabilities from an alternative repository).

Table 33. Chef recipes for remediation of one of the SVA alerts

According to the remediation plan presented in Figure 31 and the set of Chef recipes introduced in Table 33, in Table 34 we present remediation plan for the considered SVA alert SVA-E3 in terms of the sequences of Chef recipes.

<table>
<tr><th>Event</th><th colspan="5">SVA-E3</th></tr>
<tr><td rowspan="2">Action 1</td><td>Recipes</td><td colspan="4">{SVA-R1}</td></tr>
<tr><td>Result</td><td colspan="2">yes</td><td colspan="2">no</td></tr>
<tr><td rowspan="2">Action 2</td><td>Recipes</td><td colspan="2">{SVA-R2, SVA-R3, SVA-R4}</td><td colspan="2">{SVA-R5, SVA-R1}</td></tr>
<tr><td>Result</td><td>yes</td><td>no</td><td>yes</td><td>no</td></tr>
<tr><td rowspan="2">Action 3</td><td>Recipes</td><td rowspan="2">OBSERVE</td><td rowspan="2">NOTIFY</td><td colspan="2">{SVA-R2, SVA-R3, SVA-R4}</td><td rowspan="2">NOTIFY</td></tr>
<tr><td>Result</td><td>OBSERVE</td><td>NOTIFY</td></tr>
</table>

Table 34. Remediation plan in the form of Chef recipes

Step 5: Create and register Chef cookbook

When all Chef recipes for the automated SLA implementation and the SLA remediation are prepared, they need to be organized in a Chef cookbook and registered in the SPECS Chef repository (for further details about where to register cookbooks and how they are uploaded to the Chef Server, see deliverable D1.6.2).

Step 6: Create and register metadata

The last but most important step for an automated management of the developed mechanism is the creation of the mechanism's metadata representation. In this phase, the SPECS developer has to prepare a description of the mechanism behaviour, according to the SPECS security mechanism metadata (*mechanism.json*) proposed in deliverable D1.3.

Finally, this metadata description has to be registered in the SPECS SLA Platform's Service Manager component (see deliverable D1.4.1 for further details about the SLA Platform) to automate mechanism's management.