Secure Provisioning of Cloud Services
based on SLA Management

SPECS Project - Deliverable 1.5.2

Integration Examples

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.:	D1.5.2
Deliverable title:	Integration Examples
Deliverable nature:	Report
Dissemination level:	Public
Contractual delivery:	19 July 2016
Actual delivery date:	19 July 2016
Author(s):	Jolanda Modic (XLAB), Miha Stopar (XLAB)
Contributors:	Massimiliano Rak (CeRICT), Andrew Byrne (EMC), Adrian Spataru (IeAT), Silviu Panica (IeAT), Damjan Murn (XLAB), Alain Pannetrat (CSA), Giancarlo Capone (CeRICT)
Reviewers:	Umberto Villano (CeRICT), Madalina Erascu (IeAT), Stefano Marrone (CeRICT)
Task contributing to the deliverable:	T1.5
Total number of pages:	84

Executive summary

This document demonstrates implementation and testing activities associated to a set of integration scenarios introduced in deliverable D1.5.1.

In particular, this document presents:

- Overview of the integration process: We summarize the integration plan reported in the deliverable D1.5.1 to present the integration process adopted in the project. Moreover, the continuous integration approach and tools are reported, and the testing activities (as defined in T4.5) are discussed. We also introduce the template used to describe deployment of each integration scenario.
- Integration examples: We present a list of integration examples (i.e., deployed integration scenarios) that is based on the set of integration scenarios defined in D1.5.1.
- Integration and system testing: We summarize activities related to the integration and system testing as defined in T4.5. We report results of the security assessment of the SPECS applications and security review performed for the entire SPECS framework, and we elaborate on the data protection in SPECS.

Table of contents

Deliverable information 2
Executive summary 3
Table of contents 4
Index of figures 5
Index of tables 6
1. Introduction 8
2. Relationship with other deliverables 9
3. Integration process 10
3.1. Integration plan 10
3.2. Continuous integration 12
3.3. Integration example template 14
4. Integration tests 15
4.1. SPECS core components 15
4.2. SPECS applications 28
5. Integration and system testing 32
5.1. Performance and scalability analysis of the SLA Platform 32
5.1.1. Performance goals 33
5.1.2. Workload modelling 34
5.1.3. Testing environment and analysis of results 36
5.2. Security review of the SPECS framework 37
5.3. Security assessment of the SPECS application 40
5.4. Data regulation in SPECS 42
5.4.1. Personal data in SPECS 43
5.4.2. Benefits of SPECS in terms of information security 43
5.4.3. Other benefits of the SPECS platform 44
5.4.3.1. Data location control 44
5.4.3.2. Accountability 44
6. Conclusions 45
Bibliography 46
Appendix 1. SPECS prototypes 48
Appendix 2. SPECS artifacts in deliverables 50
Appendix 3. Integration user guide 51
Appendix 4. Performance analysis of the SLA Platform and the default SPECS application 53
Appendix 5. SPECS application penetration testing 60
Appendix 6. Threat catalogue 64
Appendix 7. Results of the security review 74

Index of figures

Figure 1. Relationship with other deliverables 9
Figure 2. SPECS architecture 13
Figure 3. SPECS Platform performance analysis methodology 33
Figure 4. Synthetic Workloads 35
Figure 5. Performance evaluation testing environment 36
Figure 6. Security assessment analysis 41

Index of tables

Table 1. Integration of SPECS core components 10
Table 2. Integration of SPECS applications 11
Table 3. Integration example template 14
Table 4. Integration example *Core-A1* 15
Table 5. Integration example *Core-B1* 16
Table 6. Integration example *Core-AB1* 17
Table 7. Integration example *Core-AB2* 17
Table 8. Integration example *Core-AB3* 18
Table 9. Integration example *Core-AB4* 19
Table 10. Integration example *Core-C1* 19
Table 11. Integration example *Core-C2* 20
Table 12. Integration example *Core-ABC1* 21
Table 13. Integration example *Core-ABC2* 21
Table 14. Integration example *Core-D1* 22
Table 15. Integration example *Core-CD1* 23
Table 16. Integration example *Core-ABCD1* 24
Table 17. Integration example *Core-ABCD2* 24
Table 18. Integration example *Core-ABCD3* 25
Table 19. Integration example *Core-ABCD4* 25
Table 20. Integration example *Core-ABCD5* 26
Table 21. Integration example *Core-ABCD6* 26
Table 22. Integration example *Core-ABCD7* 27
Table 23. Integration example *Core-ABCD8* 27
Table 24. Integration example *Core-ABCD9* 28
Table 25. Integration example *App-A1* 28
Table 26. Integration example *App-A2* 29
Table 27. Integration example *App-A3* 29
Table 28. Integration example *App-A4* 30
Table 29. Integration example *App-E1* 30
Table 30. Integration example *App-F1* 31
Table 31. Deliverables reporting performance tests 35
Table 32. Testing environment VM specifications 37
Table 33. SPECS checklist security areas 38
Table 34. Security review results 38
Table 35. Security review results per security category 39
Table 36. Existing threats and declared risk rating 41
Table 37. Security threats per component 42
Table 38. SLA Manager user profiles for performance tests 53
Table 39. Service Manager user profiles for performance tests 53
Table 40. Metric Catalogue user profiles for performance tests 53
Table 41. Interoperability Layer user profiles for performance tests 53
Table 42. SPECS application user profiles for performance tests 54
Table 43. Performance results for the SLA Manager 55
Table 44. Performance results for the Service Manager 56
Table 45. Performance results for the Metric Catalogue 56
Table 46. Performance results for the Interoperability Layer 57

Table 47. Performance results for the default SPECS application 59

1. Introduction

This document presents the technical aspects associated to the SPECS integration activities. In particular, we summarize the SPECS integration plan defined in deliverable D1.5.1, introduce the methodology and tools used in the integration task, and present a template with which all reported integration tests are conformant with.

In SPECS we use Bitbucket for storing the code for all the developed software (all code developed in SPECS is available on our Bitbucket account [19]), Atlassian Bamboo [3] for automatizing integration tests on a dedicated integration machine on partner IeAT cluster (its Dashboard is available at [4]), and Gatling [11] for conducting performance tests. The software developed in SPECS is uploaded to different Bitbucket projects under our Bitbucket account. Every time a developer commits a change, Bamboo compiles the code and produces new binary artifacts, the Bamboo integration plan updates the integration machine on IeAT cluster, and starts up all integration tests.

Note that all integration tests are thought to be easily repeatable and completely automated. Moreover, the integration process can be reused on every testbed in order to verify the correctness of implementation.

The reported integration tests, which have been executed for verifying the correctness of implementation of the SPECS behaviour, are divided into two sets. One set includes core components and the other set includes SPECS applications, namely the Secure Web Container, the Metric Catalogue, and the Security Reasoner. The first two have been introduced in deliverable D5.1.3, and the last one has been presented in deliverable D2.3.1. For Secure Storage application, refer to deliverable D5.2.2, for the ngDC application to deliverable D5.3, and for the AAAaaS application to deliverable D5.4.

In deliverable D4.5.2 we defined the non-functional testing approach that would be adopted on the project level. In this document we further elaborate on the methodologies for the performance and security evaluation of the developed software. We also present results for the SLA Platform and the default SPECS application, whereas for other modules we report results in dedicated deliverables (for Negotiation module in D2.3.2, for the Monitoring module in D3.4.2, and for the Enforcement module in D4.5.3).

The document is structured as follows. After a brief analysis on associated deliverables that provide an input or serve as an output in Section 2, we elaborate in the SPECS integration process in Section 3. In Section 4 we report integration tests. The performance and security evaluation methodologies are presented in Section 5, where the data protection in SPECS is also discussed. A brief summary of results, in Section 6, concludes the document.

2. Relationship with other deliverables

Deployment of the SPECS integration scenarios depends not only on the definition of scenarios itself, but also on development aspects of all components that need to be integrated. Therefore, activities associated to the integration and testing are based on a large set of deliverables as depicted in Figure 1.

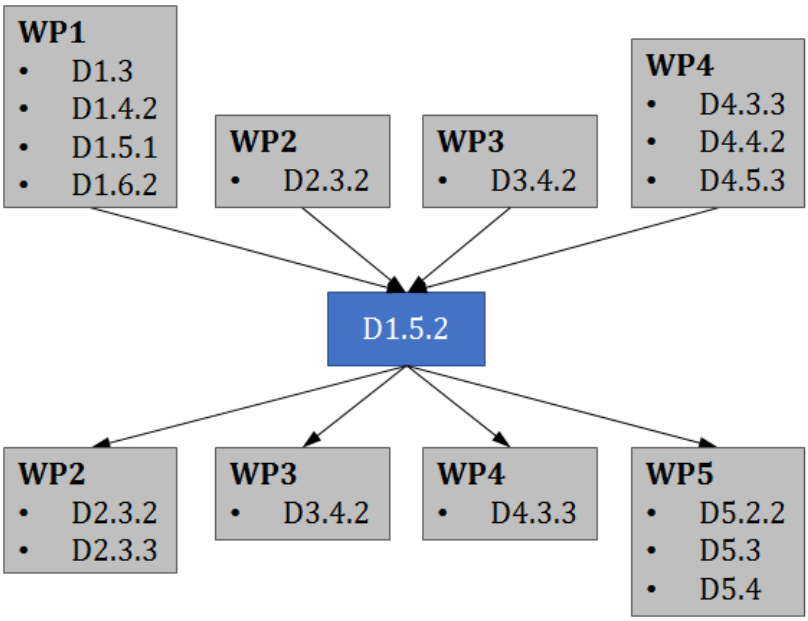


Figure 1. Relationship with other deliverables

Integration scenarios defined in D1.5.1 are implemented with accordance to the specifications of the SPECS testbed reported in D1.6.2, module interaction protocols defined in D1.3, prototypes of the components of the Vertical Layer described in D1.4.2 and D4.4.2, and prototypes of the Negotiation module, Monitoring module, and Enforcement module, described in D2.3.x, D3.4.2, and D4.3.3, respectively.

Testing activities are conducted as defined in the deliverable D4.5.3.

Feedback from the integration and testing activities are provided to the developers of the core modules for which the final prototypes are presented in deliverables D2.3.x, D3.4.2, and D4.3.3, and to the developers of the validation applications demonstrated in deliverables D5.2.2, D5.3, and D5.4.

3. Integration process

This section presents the technical details of the SPECS integration process. We summarize the SPECS integration plan and associated scenarios defined in the deliverable D1.5.1 (Section 3.1), discuss the collaborative platforms and the technological choices for our integration approach (Section 3.2), and introduce the template used to present the deployment plans for the defined integration scenarios (Section 3.3).

3.1. Integration plan

We split integration activities into two parts. First (as seen in Table 1) we plan integration of core components to enable the management of SLAs (i.e., to enable the negotiation, implementation, monitoring, and remediation processes), then (as depicted in Table 2) we organize activities to integrate core components with security mechanisms to develop a variety of SPECS applications with which we offer secure cloud services through SLAs.

Integration Scenario	Default SPECS App.	SLAP			NEG			ENF				MON							VL			
		SLA Manager	Service Manager	Interoperability Layer	SLO Manager	Supply Chain Manager	Security Reasoner	Planning	Implementation	Diagnosis	RDS	Event Hub	Event Aggregator	MoniPoli Filter	SLOM Exporter	CTP	Event Archiver	Nmap	Auditing	Security Tokens	Credential Service	User Manager
Core-A1		x			x																	
Core-B1			x		x	x		x														
Core-AB1		x	x		x	x		x														
Core-AB2		x	x		x	x		x	x													
Core-AB3		x	x		x	x		x	x			x		x								
Core-AB4	x	x	x		x	x		x	x			x		x								
Core-C1												x	x	x	x		x					
Core-C2												x	x	x	x	x	x					
Core-ABC1		x	x		x	x		x	x			x	x	x	x	x	x					
Core-ABC2	x	x	x		x	x		x	x			x	x	x	x	x	x					
Core-D1								x	x	x	x											
Core-CD1								x	x	x	x	x	x	x	x	x	x					
Core-ABCD1		x	x		x	x		x	x	x	x	x	x	x	x	x	x					
Core-ABCD2		x	x		x	x	x	x	x	x	x	x	x	x	x	x	x					
Core-ABCD3		x	x		x	x	x	x	x	x	x	x	x	x	x	x	x			x		
Core-ABCD4		x	x		x	x	x	x	x	x	x	x	x	x	x	x	x			x		x
Core-ABCD5		x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x			x		x
Core-ABCD6		x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x			x	x	x
Core-ABCD7		x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x		x	x	x	x
Core-ABCD8		x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
Core-ABCD9	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

Table 1. Integration of SPECS core components

Integration Scenario	SLAP	NEG	SM								APP					
	Metric Catalogue	Security Reasoner	WebPool	TLS	SVA	DoS	AAA	DBB	E2EE	ViPR	Secure Web Container	Secure Storage	ngDC	AAAaaS	Metric Catalogue	Security Reasoner
App-A1			x								x					
App-A2			x	x							x					
App-A3			x	x	x						x					
App-A4			x	x	x	x					x					
App-B1								x	x			x				
App-C1										x			x			
App-D1							x			x				x		
App-D2							x	x	x	x				x		
App-E1	x														x	
App-F1		x														x

Table 2. Integration of SPECS applications

As shown in Table 1, we integrate SPECS core components one by one, in an order that enables separate phases of the SLA life cycle. First (in scenarios *Core-A1* and *Core-B1*) we integrate components that orchestrate the basic version of the SLA (re)negotiation phase (without the SLA evaluation and ranking). The SLA implementation phase is enabled when scenarios *Core-ABx* are deployed. Monitoring steps are covered with scenarios *Core-Cx* and the entire flow up to (inclusive) the SLA monitoring phase is covered with scenarios *Core-ABCx*. The last phase of the SLA life cycle, namely the SLA remediation, is enabled with scenarios *Core-D1* and *Core-CD1*. Afterwards (as defined with scenarios *Core-ABCDx*) we gradually integrate the remaining components of the core SPECS architecture that orchestrate ranking of SLAs (Security Reasoner), monitor components and secure communication among them (Nmap, Security Tokens), manage credentials and user registration (Credential Service, User Manager), enable easier interoperability (Interoperability Layer), and provide logging functionalities (Auditing). The final core scenario *Core-ABCD9* integrates all core components with the default SPECS application. The deployment details for these scenarios are presented in Section 4.1.

In order to develop specific SPECS applications that form the SPECS solution portfolio (introduced in D6.2.2), we integrate the default SPECS application with security mechanisms as shown in Table 2.

The Secure Web Container application (introduced in the deliverable D5.1.3) that offers pools of virtual machines enhanced with some security features is developed/tested according to the plan defined with scenarios *App-Ax*. The deployment details for this scenario are discussed in Section 4.2.

The development of the Secure Storage and the ngDC applications that offer secure cloud storage in different usage contexts is defined with integration scenarios *Core-B1* and *Core-C1*, respectively. The integration tests are reported in deliverables D5.3 for the ngDC application and D5.2.2 for the Secure Storage application.

Integration scenarios *App-Dx* present the testing of the integration of mechanisms with the AAAaaS application. Further details about the application itself and the integration tests are available in the deliverable D5.4.

The Metric Catalogue application that manages the data for security metrics is associated to the integration scenario *App-E1* which is further discussed in Section 4.2.

The final integration scenario *App-F1* presents the development of the Security Reasoner application that offers comparison and ranking of cloud service providers. Further deployment details for the associated integration scenario are reported in Section 4.2.

3.2. Continuous integration

This section presents the technical aspects of the SPECS integration process. It demonstrates the organization of the Bitbucket repositories and provides details of the integration environment (i.e., the tools effectively used in the integration process).

The architecture of the SPECS framework, briefly summarized in the deliverable D1.5.1 and depicted in Figure 2, is highly modular. Consequently, the project's Bitbucket account has many repositories with different contents, from code for components to recipes for automated management of components (Chef cookbooks[1]) and other artifacts (e.g., data models). To organize repositories and simplify the integration process, we created three different projects; the first (SPECS) hosts the repository with the code of the components, the second (SPECSlegacy) hosts the code of old components that are no more supported, while the latest (SPECSintegration) hosts the code used for integration tests and performance analysis. Further details about the account are presented in deliverable D7.1.2. For what regard the components code, we adopted the following naming convention:

- **specs-core-***module_name-component_name*: For all components of the core modules where *module_name* is either
  - negotiation,
  - monitoring,
  - enforcement,
  - sla_platform,
  - enabling_platform, or
  - vertical_layer.
- **specs-mechanism-***module_name-component_name*: For all components of the SPECS security mechanisms where *module_name* can either be
  - *enforcement* or
  - *monitoring*,

  depending on the type of the component.
- **specs-utility-***component_name*: For the data models and the components of the vertical layer.
- **specs-app-***application_name*: For all SPECS applications.

[1] As discussed in D4.2.2, all automated deployment and management activities are orchestrated by Chef [1].

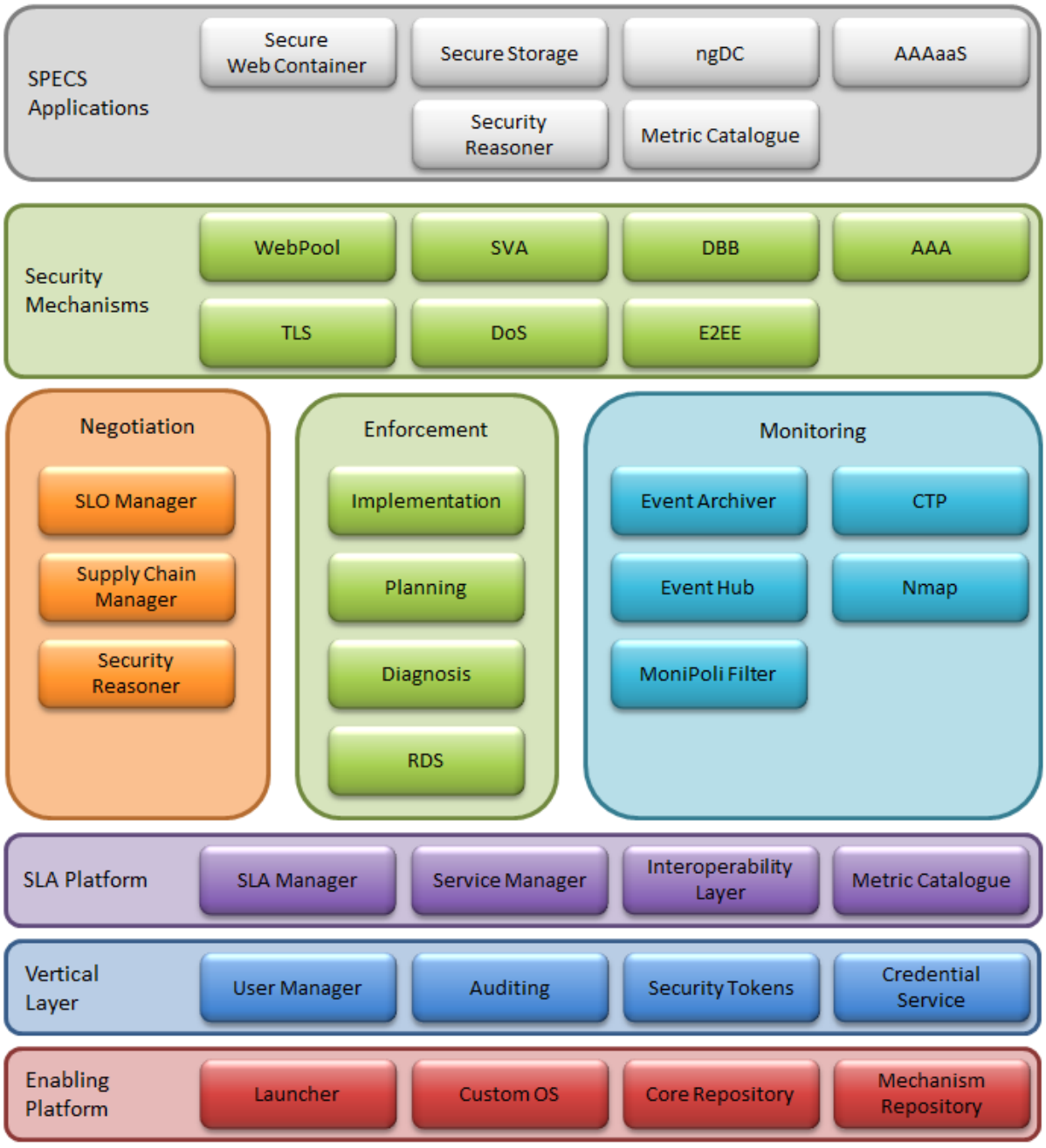


Figure 2. SPECS architecture

In Appendix 1 we present links to the project's Bitbucket repositories and web sites where the interested reader can find the code and the associated unit tests for the core components, the security mechanisms, and the developed SPECS applications. Moreover, the Appendix 1 also reports about the deliverables in which the interested reader can find the design and the implementation details of all SPECS artifacts.

The runtime environment and the supporting infrastructure that hosts the SPECS services (the SPECS Enabling Platform) and serves as the platform for the integration testing, is presented in the deliverable D1.6.2 and available on the infrastructure [2] of the project partner IeAT.

As discussed in deliverable D4.5.2, the continuous integration approach has been adopted in the project. In order to continuously assess the integration of the core components and the security mechanisms of the SPECS framework and validate that the built processes are successfully running, we use the Atlassian Bamboo server [3] which uses the Maven tool [5] for automatically building components and running unit tests. For the project's Bamboo Dashboard, please see [4].

For each integration scenario defined in deliverable D1.5.1 and reported in Table 1, we have one Bamboo deployment plan. The associated integration tests, further elaborated in Section 4.1, are available on the Bitbucket [7]. In Appendix 3 we report the details of the integration process (setting up a Bitbucket repository for tests, setting up Bamboo build plans, setting up Bamboo deployment projects, and running the integration tests).

In order to verify the correctness of the developed SPECS applications with different sets of inputs, we use Selenium [6], which is a software testing framework for web applications. All integration tests (elaborated in Section 4.2), which are used to verify correctness of the application implementation, are available on the Bitbucket [7].

3.3. Integration example template

In Table 3 we introduce a template used in Section 4 for presenting the details for each integration example. Each integration example has the same ID and the description as the associated integration scenario. For the sake of completeness we also report the list of involved SPECS artifacts. Similarly as in all prototype deliverables, we also report for each integration test the defined inputs, expected results, and the actual results. If any comments are needed for clarification, they are reported at the bottom of the table. For each integration test we also provide the link to its location on Bitbucket.

Example ID		*The ID of the integration example (the same as the ID of the associated integration scenario).*
Description		*A natural language description of the integration example outlining the roles of the involved artifacts.*
Link		*Link to the integration test.*
Core components	**SLAP**	*A list of integrated artifacts.*
	NEG	
	ENF	
	MON	
	VL	
Security mechanisms		
SPECS applications		
Inputs		*Defined inputs.*
Expected results		*Expected results of the integration test with respect to the defined inputs.*
Outputs		*Actual results of the integration test.*
Comments		*Comments, if needed, explaining the inputs, expected results or the outputs.*

Table 3. Integration example template

4. Integration tests

This section presents the deployment details for the integration scenarios defined in D1.5.1. First, tests associated to the integration of SPECS core components are presented (Section 4.1), then tests for the SPECS applications (Secure Web Container, Metric Catalogue, and Security reasoner) are reported (Section 4.2). We use the template introduced in Section 3.3.

Note that the integration examples (deployment details of the integration scenarios) for the Secure Storage, ngDC, and the AAAaaS applications are presented in dedicated deliverables D5.2.2, D5.3, and D5.4, respectively.

For further details about the SPECS flow (i.e., the SPECS framework's orchestration of the SLA life-cycle) and the APIs see deliverables D1.1.3 and D1.3, respectively.

4.1. SPECS core components

As defined in D1.5.1, the first integration test verifies the behaviour of the flow orchestrated by the SLA Manager and the SLO Manager components. With this test we verify two steps. First, when an End-user starts the negotiation process, the SLO Manager has to retrieve all SLA Templates that can be offered to the End-user. Note that each service offered by SPECS is negotiated on the basis of an individual SLA Template. Second, when the End-user selects the preferred security service, the SLO Manager has to select the associated SLA Template, customize it with the basic information such as the agreement name and context data (e.g., agreement initiator, agreement responder, service provider, expiration date, etc.), and store it in the database of the SLA Manager. The details of the test are reported in Table 4.

Example ID		*Core-A1*
Description		This scenario integrates the SLA Manager component (SLA Platform) and the SLO Manager component (Negotiation module) which provide basic functionalities for the creation and management of SLAs.
Link		https://bitbucket.org/specs-team/specs-integration-test-corea
Core components	**SLAP**	SLA Manager
	NEG	SLO Manager
	ENF	/
	MON	/
	VL	/
Security mechanisms		/
SPECS applications		/
Inputs		Based on a WSAG template (NIST or CCM) stored in the SLO Manager, this scenario is tested executing all the API calls (defined in deliverable D1.3) that imply the interaction between the SLO Manager and the SLA Manager.
Expected results		The communication between the components is correctly done. When the POST API is called on the SLO Manager to create a new SLA and, at the same time, to receive the SLA Template, a new SLA has to be stored into the SLA Manager.
Outputs		All points defined above were successfully accomplished and the results were all as expected.
Comments		/

Table 4. Integration example *Core-A1*

When the End-user selects the desired cloud service and the preferred security features to be enforced on top of it, and the SLO Manager customizes the SLA Template with this elicited information, the Supply Chain Manager orchestrates the generation of associated feasible supply chains. With the next integration test, described in Table 5, we verify whether the components involved in this process parse the inputting SLA Template correctly and correctly build all of the associated supply chains.

Example ID		*Core-B1*
Description		This scenario integrates the Service Manager component (SLA Platform), the SLO Manager and the Supply Chain Manager (Negotiation module), and the Planning component (Enforcement module). The Supply Chain Manager component (which depends on the SLO Manager) prepares the input and triggers the Planning component to build supply chains according to the SLA Template and the information provided by the Service Manager.
Link		https://bitbucket.org/specs-team/specs-integration-test-coreb
Core components	**SLAP**	Service Manager
	NEG	SLO Manager, Supply Chain Manager
	ENF	Planning
	MON	/
	VL	/
Security mechanisms		/
SPECS applications		/
Inputs		Based on a WSAG template (NIST or CCM) stored on the SLO Manager, this scenario is tested executing all the API calls (defined in D1.3) that imply the interaction among the Service Manager, SLO Manager, Supply Chain Manager, and Planning.
Expected results		The communication among the components is correctly done. When the POST API is called on the SLO Manager to receive a list of SLA Offers, all the components are correctly involved in the communication.
Outputs		All points defined above were successfully accomplished and the results were all as expected.
Comments		/

Table 5. Integration example *Core-B1*

By integrating the components of the previous two integration examples, we enable the complete negotiation process (the basic version of it, which does not include the SLA Offer ranking). With the test described in Table 6 we verify whether the inputting End-user's requirements result in a correct set of (unranked) SLA Offers.

Example ID		*Core-AB1*
Description		This scenario integrates the *Core-A1* and *Core-B1* scenarios. Involved artifacts enable the complete negotiation phase (the basic version without ranking SLA Offers).
Link		https://bitbucket.org/specs-team/specs-integration-test-coreab
Core components	**SLAP**	SLA Manager, Service Manager
	NEG	SLO Manager, Supply Chain Manager
	ENF	Planning
	MON	/

	VL	/
Security mechanisms		/
SPECS applications		/
Inputs		SLA template with WebPool and SVA capabilities, security mechanisms WebPool and SVA.
Expected results		Supply chains are successfully and correctly created corresponding to SLOs specified in the SLA template, SLA offers are successfully created, selected SLA offer is accepted and other SLA offers are deleted, SLA is signed corresponding to accepted SLA offer, SLA state is set to *signed*.
Outputs		Generated supply chains, SLA offers, signed SLA.
Comments		All points defined above were successfully accomplished and the results were all as expected.

Table 6. Integration example *Core-AB1*

When the integration test associated to the scenario *Core-AB1* succeeds, we integrate the Implementation component. With the test described in Table 7 we verify whether a signed SLA is correctly implemented. We verify (i) whether the SLA is correctly translated into an implementation plan and (ii) whether the implementation plan is correctly executed (i.e., whether all the resources specified in the SLA are automatically acquired and whether the security mechanisms needed to enforce and monitor the SLA are automatically deployed and configured).

Example ID		*Core-AB2*
Description		This scenario extends the *Core-AB1* integration scenario with the Implementation component (Enforcement module) responsible for the acquisition and configuration of cloud resources.
Link		https://bitbucket.org/specs-team/specs-integration-test-coreab
Core components	**SLAP**	SLA Manager, Service Manager
	NEG	SLO Manager, Supply Chain Manager
	ENF	Planning, Implementation
	MON	/
	VL	/
Security mechanisms		/
SPECS applications		/
Inputs		SLA template with WebPool and SVA capabilities, security mechanisms WebPool and SVA.
Expected results		Negotiation finishes successfully, planning activity and implementation activity finish successfully, planning activity state is set to *active*, implementation plan is created correctly corresponding to SLOs specified in the SLA, VMs are provisioned successfully, Chef recipes execute successfully and correct components are installed.
Outputs		SLA, planning activity, implementation activity, implementation plan.
Comments		All points defined above were successfully accomplished and the results were all as expected.

Table 7. Integration example *Core-AB2*

After an SLA is implemented (i.e., the resources are acquired and the service is correctly configured), the Enforcement module has to build an SLA with alerts (i.e., a list of parameters to be monitored and their thresholds) for the configuration of the MoniPoli Filter component.

The details of the test that verifies a correct behaviour of the complete SLA negotiation and implementation are reported in Table 8. Note that the MoniPoli Filter component depends on the Event Hub, thus we integrate both components.

Example ID		*Core-AB3*
Description		This scenario extends the *Core-AB2* integration scenario with the MoniPoli Filter component (Monitoring module) that is configured during the SLA implementation phase and is responsible for the identification of possible SLA alerts and violations.
Link		https://bitbucket.org/specs-team/specs-integration-test-coreab
Core components	**SLAP**	SLA Manager, Service Manager
	NEG	SLO Manager, Supply Chain Manager
	ENF	Planning, Implementation
	MON	MoniPoli Filter, Event Hub
	VL	/
Security mechanisms		/
SPECS applications		/
Inputs		SLA template with WebPool and SVA capabilities, security mechanisms WebPool and SVA.
Expected results		Negotiation, planning and implementation finish successfully, MoniPoli is configured successfully, correct MoniPoli rules are created corresponding to measurements specified in the security mechanisms and SLOs specified in the SLA, monitoring events generated by the mechanisms are routed through the Event Hub to the mock listener.
Outputs		SLA, implementation plan, monitoring events.
Comments		All points defined above were successfully accomplished and the results were all as expected.

Table 8. Integration example *Core-AB3*

When the behaviour of the SLA negotiation and implementation is verified, the components are integrated with the default SPECS application and the API calls from the application to the integrated components are tested. The details of this integration test are reported in Table 9.

Example ID		*Core-AB4*
Description		This scenario extends the *Core-AB3* integration scenario with the default SPECS Application. The involved artifacts enable the basic version of the SPECS flow up to the SLA monitoring phase (SLA negotiation and SLA implementation).
Link		https://bitbucket.org/specs-team/specs-integration-test-coreab
Core components	**SLAP**	SLA Manager, Service Manager
	NEG	SLO Manager, Supply Chain Manager
	ENF	Planning, Implementation
	MON	MoniPoli Filter, Event Hub
	VL	/
Security mechanisms		/
SPECS applications		Default SPECS Application
Inputs		The Selenium web application automated testing tool [13] was used to evaluate the complete end-to-end functionality of the SPECS application.
Expected results		All paths through the SPECS application are valid. The full set of

	options available is covered.
Outputs	Four test sequences were applied covering 56 individual tests (verifying options available, buttons clickable, appropriate pages loads, etc.). All tests passed.
Comments	Some of the tests required editing the web content to include HTML "id" tags. This enabled specific objects on the web application to be clickable.

Table 9. Integration example *Core-AB4*

The next group of scenarios aims at verifying correctness of the behaviour during the SLA monitoring phase. As discussed in deliverable D3.3 (design of the Monitoring module), the Event Hub collects all monitoring data received by Monitoring Adapters hosted on cloud resources (VMs). The collected monitoring data is then aggregated, archived, and filtered. The Monitoring Policy Filter component (MoniPoli) compares the data to the rules specified according to the signed SLAs. If any deviation is detected, the event is notified to the Enforcement module. The next two tables present tests with which we verified correctness of implementation of the monitoring behaviour. The first one verifies the monitoring process, and the later one additionally tests integration of the CTP Adapter which exports collected monitoring data.

Example ID		*Core-C1*
Description		This scenario integrates the components of the Monitoring module that enable collecting, aggregating, filtering and archiving events, and notifying possible SLA alerts and violations to the Enforcement module.
Link		https://bitbucket.org/specs-team/specs-integration-test-corec
Core components	**SLAP**	/
	NEG	/
	ENF	/
	MON	Event Hub, MoniPoli Filter, Event Aggregator, SLOM Exporter, Event Archiver
	VL	/
Security mechanisms		/
SPECS applications		/
Inputs		Based on a submitted SLA, a group of monitoring agents are simulated and start to feed the Event Hub with events (randomizing alerts/violations and expected events); a diagnosis mock-up is started to evaluate the notifications generated by the MoniPoli.
Expected results		Each event is: (1) routed to the Event Archiver for data archival, (2) filtered by the MoniPoli, and (3) in case of deviations notifications are generated to the Diagnosis component. At the end of the simulation for each event generated, we verify if it was successful archived, filtered by the MoniPoli, and that the corresponding notification was generated correctly.
Outputs		All points defined above were successfully accomplished and the results were all as expected.
Comments		The simulators and the mock-up service had to be custom built.

Table 10. Integration example *Core-C1*

Example ID		*Core-C2*
Description		This scenario extends the *Core-C1* scenario with the CTP component (Monitoring module), which exports monitoring data relevant to the End-user to the SPECS Application. The set of integrated components enables all monitoring functionalities.
Link		https://bitbucket.org/specs-team/specs-integration-test-corec
Core components	**SLAP**	/
	NEG	/
	ENF	/
	MON	Event Hub, MoniPoli Filter, Event Aggregator, SLOM Exporter, Event Archiver, CTP
	VL	/
Security mechanisms		/
SPECS applications		/
Inputs		Based on a submitted SLA, a group of monitoring agents are simulated and start to feed the Event Hub with events (randomizing alerts/violations and expected events); a diagnosis mock-up is started to evaluate the notifications generated by the MoniPoli.
Expected results		Each event is: (1) routed to the Event Archiver for data archival, (2) filtered by the MoniPoli, (3) in case of deviations notifications are generated to the Diagnosis component, (4) a new SLA notification is sent to the CTP, and (5) each event related to the SLA is registered in the CTP. At the end of the simulation for each event generated, we verify if it was successful archived, filtered by the MoniPoli, and that the corresponding notification was generated correctly. Moreover, we test if the generated SLAs and their corresponding events were correctly registered into the CTP.
Outputs		All points defined above were successfully accomplished and the results were all as expected.
Comments		/

Table 11. Integration example *Core-C2*

With the integration tests above, we have separately verified the correctness of implementation of the SLA negotiation, SLA implementation, and SLA monitoring. In the tables below, we integrate all components and test the entire flow. The first test includes core components involved in the mentioned processes, and the later scenario integrates the default SPECS application to evaluate the complete end to end functionality of the SPECS application.

Example ID		*Core-ABC1*
Description		This scenario integrates the *Core-AB3* and *Core-C2* scenarios. Involved artifacts enable the basic SLA negotiation (without ranking SLA Offers), SLA implementation, and SLA monitoring phases.
Link		https://bitbucket.org/specs-team/specs-integration-test-coreabc
Core components	**SLAP**	SLA Manager, Service Manager
	NEG	SLO Manager, Supply Chain Manager
	ENF	Planning, Implementation
	MON	Event Hub, MoniPoli Filter, Event Aggregator, SLOM Exporter, Event Archiver, CTP
	VL	/

Security mechanisms	/
SPECS applications	/
Inputs	In this scenario, all the components of the SLA Platform, Negotiation and Enforcement modules are involved to negotiate and implement an SLA. Subsequently, a group of monitoring agents are started to feed the Event Hub with events (alerts/violations and expected events). A Diagnosis mock-up is started to evaluate the notifications generated by the MoniPoli.
Expected results	The expected result related to the negotiation phase is the correct management of the SLA, from its creation to the implementation. Subsequently (according to the integration scenario *Core-C1*) each event is: routed to the Event Archiver for data archival, filtered by the MoniPoli, and in case of deviations notifications are generated to the Diagnosis component. At the end of the simulation for each event generated, we verify if it was successful archived, filtered by the MoniPoli, and that the corresponding notification was generated correctly.
Outputs	All points defined above were successfully accomplished and the results were all as expected.
Comments	/

Table 12. Integration example *Core-ABC1*

Example ID		*Core-ABC2*
Description		This scenario extends the *Core-ABC1* scenario with the default SPECS Application. The involved artifacts enable the basic version of the SLA negotiation, SLA implementation, and SLA monitoring.
Link		https://bitbucket.org/specs-team/specs-integration-test-coreabc
Core components	**SLAP**	SLA Manager, Service Manager
	NEG	SLO Manager, Supply Chain Manager
	ENF	Planning, Implementation
	MON	Event Hub, MoniPoli Filter, Event Aggregator, SLOM Exporter, Event Archiver, CTP
	VL	/
Security mechanisms		/
SPECS applications		Default SPECS Application
Inputs		The Selenium web application automated testing tool [13] was used to evaluate the complete end-to-end functionality of the SPECS application with the monitoring part included.
Expected results		All paths through the SPECS application are valid. The full set of options available is covered.
Outputs		All tests passed.
Comments		Some of the tests required editing the web content to include HTML “id” tags. This enabled specific objects on the web application to be clickable.

Table 13. Integration example *Core-ABC2*

The last part of the SLA life cycle to be tested is the SLA remediation. As discussed in deliverables D4.3.2 and D4.3.3, the Monitoring module notifies the Diagnosis component of the Enforcement module about an SLA alert/violation. The Diagnosis component has to analyse the received notification to determine the impact of the event on the affected SLA. The

RDS component later (on the basis of the analysis performed by the Diagnosis) identifies the optimal remediation plan and triggers the Implementation component to execute it. The described remediation process is verified with the integration scenarios presented in two tables below. In Table 14 we verify correctness of the implementation of the remediation process orchestrated by the Enforcement components (without including the Monitoring module) and in Table 15 we integrate the components of the Monitoring module (to verify correctness of the process of notifying monitoring events and to verify correctness of retrieving monitoring data from the Monitoring Archiver component, which is part of the diagnosis process).

Example ID		*Core-D1*
Description		This scenario integrates the Diagnosis and the RDS components (Enforcement module), which analyse monitoring events and prepare remediation plans according to the performed analysis.
Link		https://bitbucket.org/specs-team/specs-integration-test-cored
Core components	**SLAP**	/
	NEG	/
	ENF	Planning, Implementation, Diagnosis, RDS
	MON	/
	VL	/
Security mechanisms		/
SPECS applications		/
Inputs		SLA and corresponding supply chain, security mechanisms WebPool and SVA, violation notification, monitoring events.
Expected results		SLA is implemented successfully, VMs are provisioned, a mock violation notification triggers diagnosis activity, the corresponding monitoring event is evaluated and classified correctly, remediation activity is started, remediation plan is created correctly according to the violated metric, remediation actions are executed successfully, remediation Chef recipes are applied successfully, diagnosis and remediation activity finish successfully.
Outputs		Diagnosis and remediation activity, remediation plan.
Comments		All points defined above were successfully accomplished and the results were all as expected. Uses following mock objects: SLA Manager, Service Manager, Event Archiver, MoniPoli, CTP.

Table 14. Integration example *Core-D1*

Example ID		*Core-CD1*
Description		This scenario merges the *Core-C2* and *Core-D1* integration scenarios. Involved components enable the monitoring and remediation steps.
Link		https://bitbucket.org/specs-team/specs-integration-test-corecd
Core components	**SLAP**	/
	NEG	/
	ENF	Planning, Implementation, Diagnosis, RDS
	MON	Event Hub, MoniPoli Filter, Event Aggregator, SLOM Exporter, Event Archiver, CTP
	VL	/
Security mechanisms		/
SPECS applications		/

Inputs	SLA and corresponding supply chain, security mechanisms WebPool and SVA.
Expected results	SLA is implemented successfully, VMs are provisioned, MoniPoli rules are created, mechanisms are installed successfully, monitoring event sent by the mechanism on a VM to the Event Hub is passed to the MoniPoli which detects a potential violation and sends notification to the Diagnosis, the diagnosis activity is started, the corresponding monitoring event is evaluated and classified correctly, remediation activity is started, remediation plan is created correctly according to the violated metric, remediation actions are executed successfully, remediation Chef recipes are applied successfully, diagnosis and remediation activity finish successfully.
Outputs	Monitoring event, notification, diagnosis and remediation activity, remediation plan.
Comments	All points defined above were successfully accomplished and the results were all as expected.

Table 15. Integration example *Core-CD1*

With the integration test *Core-ABC1*, which verifies correctness of implementation of the SLA negotiation, SLA implementation, and SLA monitoring, and the integration test *Core-CD*, which verifies correctness of implementation of the SLA remediation, we implement the integration test that verifies the entire (basic) SPECS flow. The details are presented in Table 16.

Example ID		*Core-ABCD1*
Description		This scenario merges the *Core-ABC1* and *Core-CD1* integration scenarios, and enables the basic version of the entire SPECS flow (all steps except SLA ranking).
Link		https://bitbucket.org/specs-team/specs-integration-test-coreabcd
Core components	**SLAP**	SLA Manager, Service Manager
	NEG	SLO Manager, Supply Chain Manager
	ENF	Planning, Implementation, Diagnosis, RDS
	MON	Event Hub, MoniPoli Filter, Event Aggregator, SLOM Exporter, Event Archiver, CTP
	VL	/
Security mechanisms		/
SPECS applications		/
Inputs		In this scenario, all the components of the SLA Platform, Negotiation and Enforcement modules are involved to negotiate and implement an SLA. Subsequently, a group of monitoring agents are started to feed the Event Hub with events. In the end, a violation event is generated to involve the Diagnosis and the RDS components.
Expected results		The expected result related to the negotiation phase is the correct management of the SLA, from its creation to the implementation. Subsequently (according to the integration scenario *Core-C1*) each event is: routed to the Event Archiver for data archival, filtered by the MoniPoli and, in case of deviations, notifications are generated to the Diagnosis component. At this point, the Diagnosis component is activated and, if the SLA is violated, the RDS component in activated, too.
Outputs		All points defined above were successfully accomplished and the

	results were all as expected.
Comments	/

Table 16. Integration example *Core-ABCD1*

The remaining set of tests presents integration of components that are not crucial for the implementation of the core SPECS flow ((re)negotiation, implementation, monitoring, and remediation), but support it. In Table 17 we present integration of the Security Reasoner component, which compares different SLA Offers and ranks them according to End-user's security requirements to help the End-user to decide on the best fitting SLA Offer.

Example ID		*Core-ABCD2*
Description		This scenario extends the *Core-ABCD1* scenario by integrating the Security Reasoner component (Negotiation module). By including functionalities related to the evaluation and ranking of the SLAs, this scenario enables the complete SPECS flow.
Link		https://bitbucket.org/specs-team/specs-integration-test-coreabcd
Core components	**SLAP**	SLA Manager, Service Manager
	NEG	SLO Manager, Supply Chain Manager, Security Reasoner
	ENF	Planning, Implementation, Diagnosis, RDS
	MON	Event Hub, MoniPoli Filter, Event Aggregator, SLOM Exporter, Event Archiver, CTP
	VL	/
Security mechanisms		/
SPECS applications		/
Inputs		In this scenario, during the negotiation process, a list of SLA Offers is requested.
Expected results		The expected result is that the rank of each returned SLA Offer is equal to the rank computed querying the Security Reasoner about each SLA Offer.
Outputs		All points defined above were successfully accomplished and the results were all as expected.
Comments		/

Table 17. Integration example *Core-ABCD2*

In the next two tables (Table 18 and Table 19) we present integration of the Security Tokens mechanism and the User Manager component. The first one ensures secure communication among SPECS components, and the later one supports the authentication and authorization of SPECS users.

Example ID		*Core-ABCD3*
Description		This scenario extends the *Core-ABCD2* scenario by integrating the Security Tokens mechanism (component of the Vertical Layer), which is responsible for the security of interactions among SPECS components.
Link		https://bitbucket.org/specs-team/specs-integration-test-coreabcd
Core components	**SLAP**	SLA Manager, Service Manager
	NEG	SLO Manager, Supply Chain Manager, Security Reasoner
	ENF	Planning, Implementation, Diagnosis, RDS
	MON	Event Hub, MoniPoli Filter, Event Aggregator, SLOM Exporter, Event

		Archiver, CTP
	VL	Security Tokens
Security mechanisms		/
SPECS applications		/
Inputs		In this scenario, the interaction between SPECS components has been tested both with valid and invalid tokens.
Expected results		With the use of invalid tokens, no interaction between components is possible, so it is not possible to execute the negotiation process that needs the interaction of many components. With the use of valid tokens, the interaction between components has to be successfully, and the negotiation process has to complete.
Outputs		All points defined above were successfully accomplished and the results were all as expected.
Comments		/

Table 18. Integration example *Core-ABCD3*

Example ID		*Core-ABCD4*
Description		This scenario extends the *Core-ABCD3* scenario by integrating the User Manager component (Vertical Layer), which oversees authentication and authorization functionalities to SPECS users.
Link		https://bitbucket.org/specs-team/specs-integration-test-coreabcd
Core components	**SLAP**	SLA Manager, Service Manager
	NEG	SLO Manager, Supply Chain Manager, Security Reasoner
	ENF	Planning, Implementation, Diagnosis, RDS
	MON	Event Hub, MoniPoli Filter, Event Aggregator, SLOM Exporter, Event Archiver, CTP
	VL	Security Tokens, User Manager
Security mechanisms		/
SPECS applications		/
Inputs		In this scenario, it has been evaluated the access to the SPECS functionalities in the case of an unauthenticated user, an authenticated user with defined role privileges, and a user with full privileges.
Expected results		Accessing SPECS functionalities as an unauthenticated user must not be allowed. Accessing SPECS functionalities as an authenticated user with a defined role allows the access only to those functionalities enabled for that role. Accessing SPECS functionalities as an authenticated user with a full privileges role allows the access to all functionalities offered by SPECS.
Outputs		All points defined above were successfully accomplished and the results were all as expected.
Comments		/

Table 19. Integration example *Core-ABCD4*

The next three tests extend the integration test *Core-ABCD4* and test the integration of the Interoperability Layer component, Credential Manager mechanism, and the Auditing component. The added functionalities, used inputs, and expected and actual outputs are presented below (in Table 20, Table 21, and Table 22).

Example ID		*Core-ABCD5*
Description		This scenario extends the *Core-ABCD4* scenario by integrating the Interoperability Layer component (SLA Platform), which offers the single access point to all SPECS APIs.
Link		https://bitbucket.org/specs-team/specs-integration-test-coreabcd
Core components	**SLAP**	SLA Manager, Service Manager, Interoperability Layer
	NEG	SLO Manager, Supply Chain Manager, Security Reasoner
	ENF	Planning, Implementation, Diagnosis, RDS
	MON	Event Hub, MoniPoli Filter, Event Aggregator, SLOM Exporter, Event Archiver, CTP
	VL	Security Tokens, User Manager
Security mechanisms		/
SPECS applications		/
Inputs		In this scenario, the interaction between SPECS components has been tested using the Interoperability Layer.
Expected results		The interaction between components has to be successful.
Outputs		All points defined above were successfully accomplished and the results were all as expected.
Comments		/

Table 20. Integration example *Core-ABCD5*

Example ID		*Core-ABCD6*
Description		This scenario extends the *Core-ABCD5* scenario by integrating the Credential Service mechanism (Vertical Layer), which stores and manages SPECS Owner credentials to access the external resources.
Link		https://bitbucket.org/specs-team/specs-integration-test-coreabcd
Core components	**SLAP**	SLA Manager, Service Manager, Interoperability Layer
	NEG	SLO Manager, Supply Chain Manager, Security Reasoner
	ENF	Planning, Implementation, Diagnosis, RDS
	MON	Event Hub, MoniPoli Filter, Event Aggregator, SLOM Exporter, Event Archiver, CTP
	VL	Security Tokens, User Manager, Credential Service
Security mechanisms		/
SPECS applications		/
Inputs		In this scenario, the credentials useful to access to external resources, have been stored and associated to each external resource accessible. Then those credentials have to be retrieved to access different resources.
Expected results		The retrieved credentials must grant access to external resources.
Outputs		All points defined above were successfully accomplished and the results were all as expected.
Comments		/

Table 21. Integration example *Core-ABCD6*

Example ID		*Core-ABCD7*
Description		This scenario extends the *Core-ABCD6* scenario by integrating the Auditing component (Vertical Layer), which offers logging functionalities to all components of the SPECS framework.
Link		https://bitbucket.org/specs-team/specs-integration-test-coreabcd
Core	**SLAP**	SLA Manager, Service Manager, Interoperability Layer

components	**NEG**	SLO Manager, Supply Chain Manager, Security Reasoner
	ENF	Planning, Implementation, Diagnosis, RDS
	MON	Event Hub, MoniPoli Filter, Event Aggregator, SLOM Exporter, Event Archiver, CTP
	VL	Security Tokens, User Manager, Credential Service, Auditing
Security mechanisms		/
SPECS applications		/
Inputs		In this scenario, the activation of each SPECS component has been useful to test the log of each "operation".
Expected results		The activation and the login procedure of each SPECS component has to be properly logged.
Outputs		All points defined above were successfully accomplished and the results were all as expected.
Comments		/

Table 22. Integration example *Core-ABCD7*

With the last two core integration tests we verify correctness of implementation of the Nmap mechanism that oversees availability of SPECS components (scenario Core-ABCD8 in Table 23) and we evaluate the entire end-to-end functionality of the default SPECS application (scenario Core-ABCD9 in Table 24).

Example ID		*Core-ABCD8*
Description		This scenario extends the *Core-ABCD7* scenario by integrating the Nmap mechanism (Monitoring module), which monitors availability of internal components of the SPECS framework.
Link		https://bitbucket.org/specs-team/specs-integration-test-coreabcd
Core components	**SLAP**	SLA Manager, Service Manager , Interoperability Layer
	NEG	SLO Manager, Supply Chain Manager, Security Reasoner
	ENF	Planning, Implementation, Diagnosis, RDS
	MON	Event Hub, MoniPoli Filter, Event Aggregator, SLOM Exporter, Event Archiver, CTP, Nmap
	VL	Security Tokens, User Manager, Credential Service, Auditing
Security mechanisms		/
SPECS applications		/
Inputs		In this scenario, all the components are up and running. Then some of them become unavailable.
Expected results		If one or more internal components of the SPECS framework get unavailable, Nmap has to get the state of each of those components, and send related unavailability event to the Event Hub.
Outputs		All points defined above were successfully accomplished and the results were all as expected.
Comments		In this scenario, all the components are up and running, then some of them become unavailable.

Table 23. Integration example *Core-ABCD8*

Example ID		*Core-ABCD9*
Description		This scenario extends the *Core-ABCD8* scenario by integrating the default SPECS Application.
Link		https://bitbucket.org/specs-team/specs-integration-test-coreabcd
Core	**SLAP**	SLA Manager, Service Manager, Interoperability Layer

components	**NEG**	SLO Manager, Supply Chain Manager, Security Reasoner
	ENF	Planning, Implementation, Diagnosis, RDS
	MON	Event Hub, MoniPoli Filter, Event Aggregator, SLOM Exporter, Event Archiver, CTP, Nmap
	VL	Security Tokens, User Manager, Credential Service, Auditing
Security mechanisms		/
SPECS applications		Default SPECS Application
Inputs		The specs application has to guide the user through all the phases, and the inputs are the same as provided in the tables from integration scenarios *Core-ABCD1* to *Core-ABCD8*. The Selenium web application automated testing tool [13] was used for this evaluation.
Expected results		The same result obtained before have to be properly notified to the user on the UI.
Outputs		All points defined above were successfully accomplished and the results were all as expected.
Comments		/

Table 24. Integration example *Core-ABCD9*

4.2. SPECS applications

In the following four tables we present the development of the Secure Web Container application, i.e., integration of the SPECS framework and a particular set of security mechanisms with the default SPECS application.

The Secure Web Container application offers to cloud users virtual machines (VMs) with the WebPool security mechanism, and offers additional security guarantees like software vulnerability assessment, TLS protocol, and denial of service detection and mitigation with SVA, TLS, and DoS mechanisms, respectively.

Further details about the application have been reported in deliverable D5.1.3.

Example ID	*App-A1*
Description	This scenario integrates the Secure Web Container application with the SPECS WebPool security mechanism.
Link	https://bitbucket.org/specs-team/specs-integration-test-appa
Core components	All
Security mechanisms	WebPool
SPECS applications	Secure Web Container
Inputs	In this scenario, during the negotiation process, the WebPool security capability is chosen with different combinations of the Level of Redundancy and Level of Diversity security metrics (see WebPool section in deliverable D4.3.2).
Expected results	The acquired resources and their configurations have to be compliant with what has been defined during the SLA negotiation process related to the WebPool mechanism.
Outputs	All points defined above were successfully accomplished and the results were all as expected.
Comments	/

Table 25. Integration example *App-A1*

Example ID	*App-A2*
Description	This scenario extends the *App-A1* integration scenario with the TLS security mechanism. It integrates the Secure Web Container application with the WebPool and TLS security mechanisms.
Link	https://bitbucket.org/specs-team/specs-integration-test-appa
Core components	All
Security mechanisms	WebPool, TLS
SPECS applications	Secure Web Container
Inputs	In this scenario, during the negotiation process, a cloud service is chosen with different combinations of metrics enforced and monitored by the WebPool and the TLS mechanisms (see deliverable D4.3.2 for details about the mechanisms).
Expected results	The acquired resources and their configurations have to be compliant with what has been defined during the SLA negotiation process related to the WebPool and the TLS mechanism.
Outputs	All points defined above were successfully accomplished and the results were all as expected.
Comments	/

Table 26. Integration example *App-A2*

Example ID	*App-A3*
Description	This scenario extends the *App-A2* integration scenario with the SVA security mechanism. It integrates the Secure Web Container application with the WebPool, TLS, and SVA security mechanisms.
Link	https://bitbucket.org/specs-team/specs-integration-test-appa
Core components	All
Security mechanisms	WebPool, TLS, SVA
SPECS applications	Secure Web Container
Inputs	In this scenario, during the negotiation process, a cloud service is chosen with different combinations of metrics enforced and monitored by the WebPool, the TLS, and the SVA mechanisms (see deliverables D4.3.2 and D4.3.3 for details about the mechanisms).
Expected results	The acquired resources and their configurations have to be compliant with what has been defined during the SLA negotiation process related to the WebPool, the TLS, and the SVA mechanisms.
Outputs	All points defined above were successfully accomplished and the results were all as expected.
Comments	/

Table 27. Integration example *App-A3*

Example ID	*App-A4*
Description	This scenario extends the *App-A3* integration scenario with the DoS security mechanism. It integrates the Secure Web Container application with the WebPool, TLS, SVA, and DoS security mechanisms.
Link	https://bitbucket.org/specs-team/specs-integration-test-appa
Core components	All
Security mechanisms	WebPool, TLS, SVA, DoS
SPECS applications	Secure Web Container
Inputs	In this scenario, during the negotiation process, a cloud service is chosen with different combinations of metrics enforced and monitored

	by the WebPool, the TLS, the SVA, and the DoS mechanisms (see deliverables D4.3.2 and D4.3.3 for details about the mechanisms).
Expected results	The acquired resources and their configurations have to be compliant with what has been defined during the SLA negotiation process related to the WebPool, the TLS, the SVA, and the DoS mechanisms.
Outputs	All points defined above were successfully accomplished and the results were all as expected.
Comments	/

Table 28. Integration example *App-A4*

The last two integration tests reported in this section (Table 29 and Table 30) present the development of the Metric Catalogue application introduced in deliverable D5.1.3 and the Security Reasoner application introduced in deliverable D2.3.1.

Example ID		*App-E1*
Description		This scenario integrates the Metric Catalogue application with the Metric Catalogue component (part of the SLA Platform).
Link		https://bitbucket.org/specs-team/specs-integration-test-appe
Core components	**SLAP**	Metric Catalogue
Security mechanisms		/
SPECS applications		Metric Catalogue
Inputs		The Selenium web application automated testing tool [13] was used to evaluate the complete end to end functionality of the Metric Catalogue application and its interaction with the Metric Catalogue component.
Expected results		All paths through the Metric Catalogue application are valid. The full set of options available is covered.
Outputs		All points defined above were successfully accomplished and the results were all as expected.
Comments		Some of the tests required editing the web content to include HTML “id” tags. This enabled specific objects on the web application to be clickable.

Table 29. Integration example *App-E1*

Example ID		*App-F1*
Description		This scenario integrates the Security Reasoner application with the Security Reasoner component (part of the Negotiation module).
Link		https://bitbucket.org/specs-team/specs-integration-test-appf
Core components	**NEG**	Security Reasoner
Security mechanisms		/
SPECS applications		Security Reasoner
Inputs		The Selenium web application automated testing tool [13] was used to evaluate the complete end to end functionality of the Security Reasoner application and its interaction with the Security Reasoner component.
Expected results		All paths through the Security Reasoner application are valid. The full set of options available is covered.
Outputs		All points defined above were successfully accomplished and the results were all as expected.

Comments	Some of the tests required editing the web content to include HTML "id" tags. This enabled specific objects on the web application to be clickable.

Table 30. Integration example *App-F1*

5. Integration and system testing

The testing activities aimed at analysing the non-functional aspects of the developed framework/applications in SPECS have been defined in deliverable D4.5.2. In the next subsections we discuss the details of the methodologies and present the results of performed evaluations/tests.

In particular, we present the methodology for evaluating performance and scalability aspects of the SPECS components/modules. Afterwards, we present the approach to and the results of the security review conducted on the SPECS module level and the security assessment of the SPECS applications. Finally, we discuss the data regulation in SPECS.

5.1. Performance and scalability analysis of the SLA Platform

In order to offer a clear evaluation of the performance of the SPECS framework, we created a SPECS benchmark (available at SPECS Bitbucket repository [20]) that enables us to make a detailed performance analysis of the SPECS platform. The benchmarks produce performance figures that evaluate both the SPECS applications, which models the performance perceived by customers, and each of the API offered by SPECS core modules, that enable the SPECS Owner to evaluate the overheads and limits that the SPECS solution may introduce when delivering secured services.

Performance tests aim at evaluating the capability of the system and the performance perceived by the End-user when accessing SPECS Security Services, like the Secure Web Container and/or the Secure Storage.

Section 5.1.1 focuses on the goals of the performance analysis, which we can synthesize in our evaluation of the performance limits that SPECS may introduce when offering secured services. The results, summarized in Appendix 4, demonstrate that even at the state of the art (i.e., with the Technology Readiness Level[2] (TRL) between 3 and 4), the solution offers performance that does not limit a small CSP.

Figure 3 illustrates the methodology adopted to analyse the performance of the SPECS platform. It follows the common steps of a capability planning analysis as suggested by Jain [10].

The first step of the adopted methodology, namely *Performance Goals*, consists in identifying the main goals of the performance analysis process (see 5.1.1). It looks trivial, but a good performance analysis process should have a clear goal in order to produce satisfying results. The second step, namely *Workload Modelling*, focuses on the modelling of the SPECS platform usage, in order to correctly identifying the behaviour of the overall solution (see 5.1.2). The third step, *Testing Environment*, focuses on the execution environment which heavily affects measurement, that must be repeated if the execution environment changes (see 5.1.3). The last step focuses on measurement collection and analysis of the results (see Appendix 4).

[2] http://ec.europa.eu/research/participants/data/ref/h2020/wp/2014_2015/annexes/h2020-wp1415-annex-g-trl_en.pdf

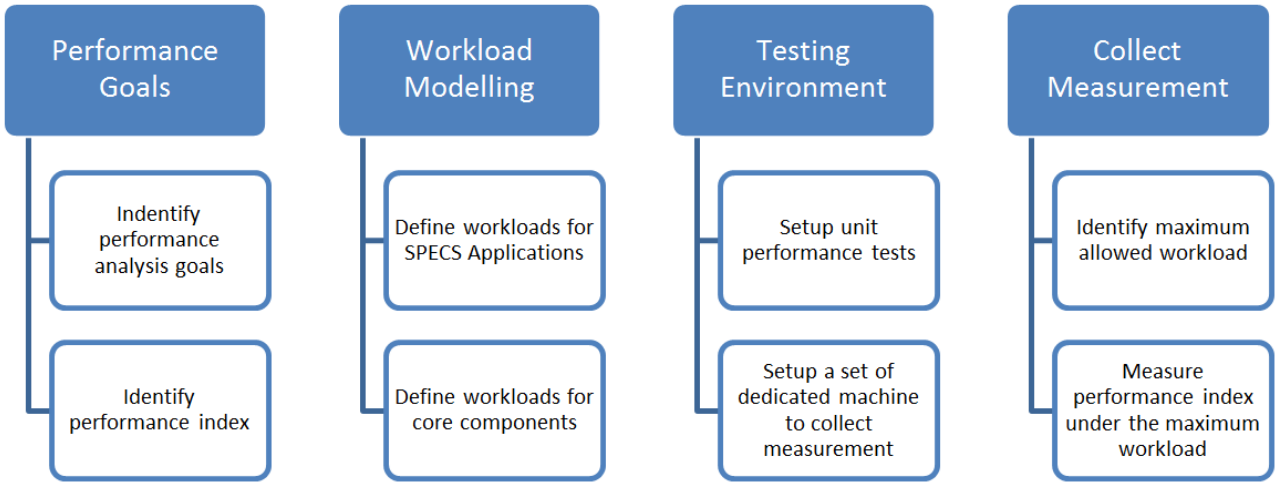


Figure 3. SPECS Platform performance analysis methodology

5.1.1. Performance goals

The goal of our tests is to evaluate what are the limits in terms of performance of the SPECS framework at the state of the art. The goal of the analysis is to evaluate if and how the adoption of the SPECS framework, as an integrated solution, affects the performance of a CSP. The main limits, that an additional layer may introduce, are the effects on the number of End-users that can concurrently use the resources secured by SPECS applications, and the time needed to reply to End-users, which affects the perception of the quality from final customers' point of view.

The same goal applies to each module of the SPECS framework; we should evaluate how their interfaces (i.e., the APIs offered by the modules; described in deliverable D1.3) behave with respect to the number of concurrent users, and what are the additional delays introduced due to services invocation.

As already outlined in the introduction of Section 5.1, the produced performance tests are a set of "SPECS benchmarks" that enable an easy evaluation of the SPECS framework on different testbeds.

According to such considerations, the target of our performance analysis are the interfaces offered by SPECS application (to evaluate the overall platform) and by each module, i.e. the REST APIs described in deliverable D1.3.

We adopted two main performance indexes:

- ***Throughput (req/s)***: the number of services executed per second. It offers a clear evaluation of the number of requests that our components are able to manage and of the capacity of the system.
- ***Response Time (ms)***: the time elapsed between the requests of a service up to the production of the result. This index evaluates the overhead introduced by the API under evaluation, and the performance perceived by End-users.

These performance indexes are evaluated for different workloads, described in the following section.

5.1.2. Workload modelling

SPECS applications offer their functionalities through dedicated web interfaces, and SPECS platform core components offer their functionalities through the REST APIs described in deliverable D1.3. This implies that all performance tests will stress HTTP layers and will have a request/response behaviour.

As outlined above, the main goal of the performance analysis is to offer a set of reusable benchmarks that can be used to correctly setup the platform in different environments. The benchmarks rely on a set of scripts that models the typical workload to which SPECS applications and core components are subject.

In order to build up the benchmark workloads, we identify a set of standard profiles composed in order to build the synthetic workloads.

Core components profiles follow the common flows of usage of the REST APIs of the SPECS modules and match the SPECS flow described in deliverable D1.3 (which represents the normal usage of the platform components).

As an example, the Service Manager component offers a REST API able to retrieve the metadata associated to security mechanisms hosted on the platform. It offers an API to retrieve the list of mechanism and an API to retrieve the metadata for a single specific mechanism. A common workload is the sequence of requests:

1. Get the list of mechanisms.
2. Get the mechanism's metadata.

One of the usage profiles for the Service Manager executes such sequence of REST API invocations. Such patterns can be simply identified for each of the core components and constitute the basic set of workloads we collected.

We build SPECS application usage profiles that represent the performance perceived by End-users, registering the application browsing and replicating the common wizard flow through our benchmark scripts.

Synthetic workloads are given generating a load of users, according to the above described profiles, varying the rate of users per second.

Figure 4 illustrates the process adopted to obtain the Synthetic Workloads: we stress the target component with a ramp of increasing users up to the limit of correct behaviour of the target component. We make this process two times, one with long-term runs and a second with short-term runs. Once we have identified the maximum number of allowed concurrent users for a component, we stress it for a fixed amount of time, measuring the Response time and the Throughput in those conditions.

The process results in a set of performance figures made of two diagrams for each user profile. The first reports the throughput and the second one reports the response time for each injected rate per second from 0 up to the maximum number of users supported by the API, according to the analysis of steps 1 and 2 of the methodology.

A detailed description of the user profiles for each component are reported in the deliverables associated to their implementation, except for the tests associated to SLA Platform components, reported in this deliverable in Appendix 4. Table 31 summarizes the components subject to tests and the deliverables reporting their performance tests.

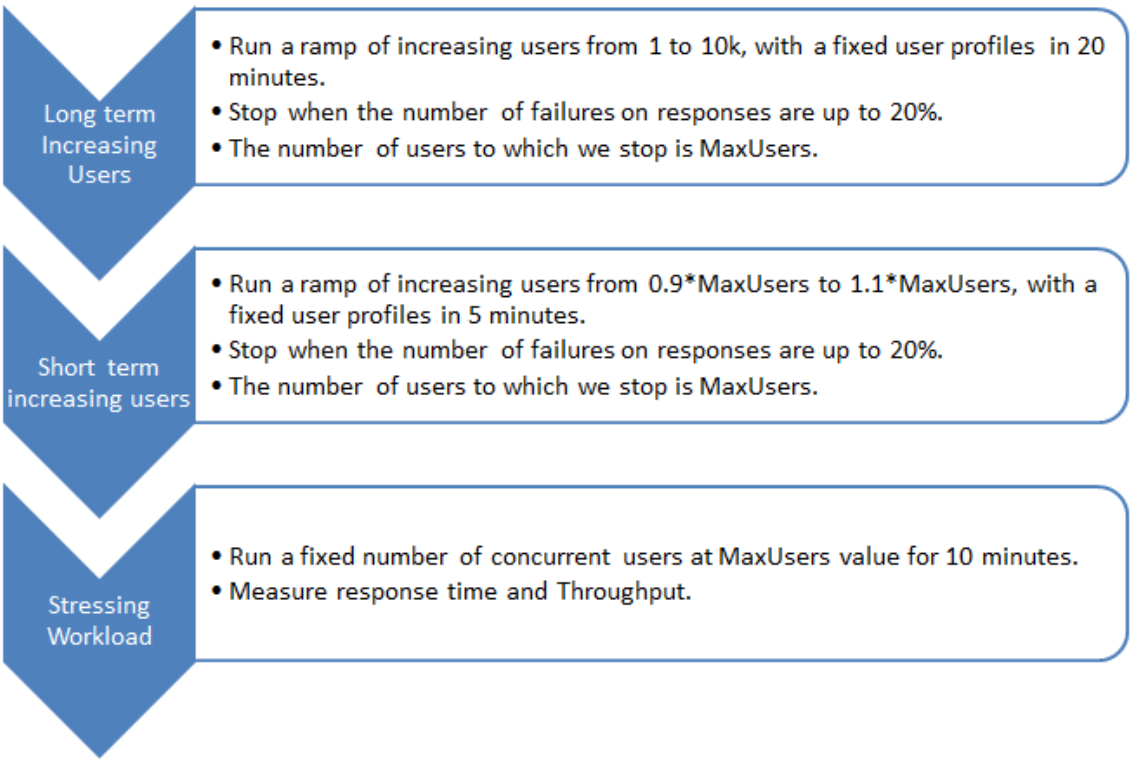


Figure 4. Synthetic Workloads

Module	Component	Deliverable (Section)
SLA Platform	SLA Manager	D1.5.2 (Appendix 4)
	Service Manager	D1.5.2 (Appendix 4)
	Interoperability Layer	D1.5.2 (Appendix 4)
Negotiation	SLO Manager	D2.3.2 (Section 3.2.4)
	Security Reasoner	D2.3.2 (Section 3.4.4)
Enforcement	Planning	D4.5.3 (Section 6.2)
	Implementation	D4.5.3 (Section 6.3)
	Diagnosis	D4.5.3 (Section 6.4)
	RDS	D4.5.3 (Section 6.5)
Monitoring	Event Hub	D3.4.2 (Section 3.2.1)
	Event Archiver	D3.4.2 (Section 3.3.3)
	MoniPoli Filter	D3.4.2 (Section 3.4.3)
Application	Secure Web Container	D1.5.2 (Appendix 4)
	Secure Storage	D5.2.2 (Section 5)
	NgDC	D5.3 (Section 6.3)
	AAAas-a-Service	D5.4 (Section 6)

Table 31. Deliverables reporting performance tests

Note that the Supply Chain Manager (a component of the Negotiation module) is not a web application, but just a Java library, directly and internally invoked by the SLO Manager component. Therefore no performance tests have been explicitly made for this component. However, since the Supply Chain Manager is a library acting as an interface between the SLO Manager and the Planning component (Enforcement module), we direct the reader to D4.5.3 (Section 6) for performance tests for the Planning component.

5.1.3. Testing environment and analysis of results

All SPECS tests run on top of the SPECS Enabling Platform, which is the default testbed, described in deliverables D1.6.1 and D1.6.2.

The testing environment, i.e. the SPECS testbed, is a university cluster with a limited dimension that well emulates a typical private cloud or a small CSP. The SPECS Platform hosted in such environment should be able to address few requests per day.

All integration tests, as described in this deliverable, run on a dedicated integration environment, which is a full running platform. We run all the scripts against the real environment, in order to verify the behaviour in the real environment. These tests, however, are limited due to the testbed capacity: in order to run the full tests in the real environment we should emulate multiple users acquiring resources on the testbed. The amount of resources available (reported in deliverable D1.6.1, about 20-30 VMs) is much less than the capability offered by the platform.

Figure 5 illustrates the execution environment adopted for the performance tests. Note that we used a dedicated VM to host the load generator, while the SPECS Platform, which is the System Under Test (SUT), is composed of two VMs: one hosting all components of the SPECS Platform and the other the Chef server. As outlined before, the tests stress the SPECS API and the default application web interface.

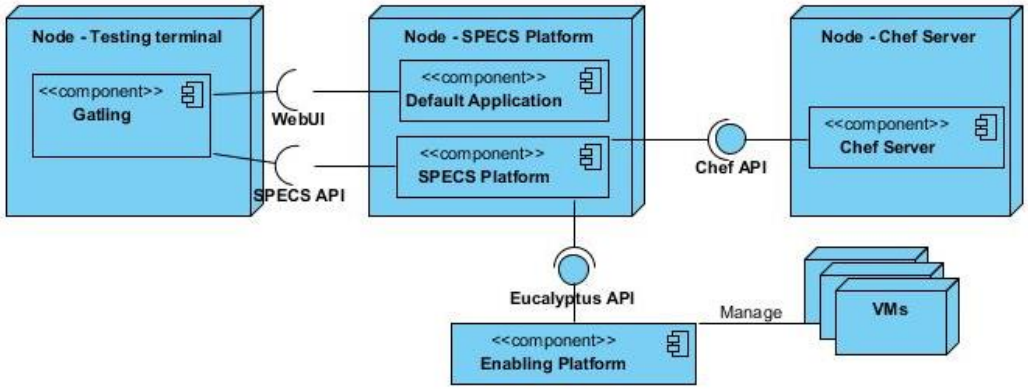


Figure 5. Performance evaluation testing environment

Table 32 summarizes the VM specifications for the three nodes reported above.

The testing terminal hosts the Gatling tool [11] that emulates the requests according to our scripts and automatically produces all reports in HTML format. All scripts are available on our SPECS Bitbucket repository [20].

VM	Description	Role
Testing Terminal	VMtype:m1.1xlarge Core: 1 RAM: 1024 HDD: 20 GB	Hosts the application that generates the load on the SPECS Platform.
SPECS Platform	VMtype:m1.1xlarge Core: 1 RAM: 1024 HDD: 20 GB	Hosts all components of the SPECS Platform and the default application.
Chef Server	VMtype:m2.2xlarge Core: 2 RAM: 2048 HDD: 9 GB	Hosts the Chef server.

Table 32. Testing environment VM specifications

The Appendix 4 contains the detailed performance figures for all the SLA Platform components and the user profiles. Our repositories contain the detailed performance results obtained for each component. Note that results of the performance analysis for other modules are reported in other deliverables as reported in Table 31.

It should be pointed out that, even considering the minimal environment exposed in Figure 5 and Table 32, the platform and the application are able to manage hundreds of users with acceptable response time (worst case measured in seconds), which respects the requirements outlined above.

It is also worth noting that the bootstrap of a VM usually is measured in minutes: a cloud customer using the SPECS infrastructure is not able to perceive the difference of the resource delivery through the platform. From the CSP perspective, the additional resource consumption is mainly due to the VM hosting the platform.

As a final consideration, the available benchmark enables us to perform a detailed capacity planning for the full SPECS Platform and, consequently, to tune up the Platform to the performance requirements that different deployments may have.

5.2. Security review of the SPECS framework

In deliverable D4.5.2 we introduced an approach to the security evaluation of the components developed in the project, which is based on the Application Security Verification Standard (ASVS 2.0) proposed by OWASP [12]. We defined a list of requirements (ordered in groups according to security areas as reported in [12]) that should be addressed during the development stage in order to assure secure software.

SPECS developers have assessed SPECS components, the testbed, and the default SPECS application. Full results are reported in Appendix 7, while Table 34 presents a summary for each layer of the SPECS architecture (as illustrated in Figure 2).

ID	Security area
A	Authentication
B	Access control
C	Malicious input handling
D	Cryptography at rest
E	Error handling and logging
F	Data protection
G	Communications security
H	HTTP security
I	Malicious controls
J	Business logic
K	Files and resources

Table 33. SPECS checklist security areas

The developers have assessed each component/testbed/application by verifying whether each requirement on the defined checklist has been covered or not (these requirements are labelled as ✓ or ×, respectively). If some requirement is not applicable to the artifact under evaluation, or the coverage of the requirement depends on the configuration, we use labels *N* and *D*, respectively. In the tables below we report and discuss (for each module) the number of requirements per each security area and present the number of implemented, not implemented, not applicable, and deployment dependent requirements for each component.

Module	**Component**	✓	×	N	D
Enabling Platform		29	19	54	4
SLA Platform	SLA Manager	33	33	34	6
	Service Manager	33	33	34	6
	Metric Catalogue	33	33	34	6
	Interoperability Layer	33	33	34	6
Negotiation	SLO Manager	25	0	81	0
	Supply Chain Manager	18	22	65	1
	Security Reasoner	33	33	34	6
Enforcement	Planning	36	30	35	5
	Implementation	36	30	35	5
	Diagnosis	36	30	35	5
	RDS	36	30	35	5
Monitoring	Event Hub	33	19	50	4
	Event Archiver	28	17	57	4
	MoniPoli Filter	52	12	40	2
	CTP	33	19	50	4
	Nmap	37	19	46	4
Vertical Layer	Auditing	36	30	35	5
	Security Tokens	36	30	35	5
	Credential Service	38	30	33	5
	User Manager	33	33	34	6
SPECS application		33	33	34	6
EMC Testbed		38	37	25	6

Table 34. Security review results

As already outline, the main goal of the security review was to identify the main security issues that may still be and that need to be addressed in order to move the solution to a higher TRL level.

The analysis outlines that the solution is compatible with the declared TRL3/TRL4 levels: in most of the cases we have more positive answers respect to negative ones. Few replies depend on configurations to be adopted, which means that the associated requirements must be addressed in case of deployment in a different environment.

The status of the components is homogeneous: most of them have a number of positive answers between 33 and 38, with the only relevant exception being the Supply Chain Manager component (having only 18 requirements implemented). It has a higher number of Not applicable (N) replies, due to its implementation (the Supply Chain Manager is a library).

Table 35 summarizes the global results per security area. Access Control is almost correctly enforced and with only a few additional improvements, the remaining issues should be solved before moving any component to the TRL5.

ID	Security area	✓	×	*N*	*D*
A	Authentication	102	4	272	13
B	Access control	158	29	43	0
C	Malicious input handling	118	143	38	0
D	Cryptography at rest	9	1	151	0
E	Error handling and logging	79	50	193	0
F	Data protection	44	36	35	0
G	Communications security	20	39	75	73
H	HTTP security	62	47	6	0
I	Malicious controls	105	62	48	15
J	Business logic	70	104	56	0
K	Files and resources	49	61	28	0

Table 35. Security review results per security category

The main issues are related to the handling of malicious input (class C). At the state of the art, the applications are tested in a laboratory. In order to expose an application to public, a set of controls needs to be added to verify correctness of the inputs. This is a relevant action to take into account to move to TRL 8 (that implies an access from public).

Similarly, the main limit in classes E and F are due to the need of additional controls against possible malicious access to the application: logs should be protected and monitoring of anomaly accesses to data should be enforced in a production environment. These issues should be solved before moving to TRL5, which implies access to a relevant environment, where real information can be stored.

Classes G and H, affecting the communication channels used, imply that before moving to a relevant environment, the detailed verification of protection of every communication channel should be ensured. At the state of the art, due to debugging reason, we have open access to some services. Moreover, a detailed assessment of the privileges assigned to each component should be made (requirement SC73) before moving to TRL5.

Class J (Business logic) and class K (Files and resources) are the security areas that need major improvements in order to protect any possible behavior of the application. These issues will become critical when moving to untrusted environments (i.e. TRL>7).
The security review helped us to identify the main actions needed to move the solution up to TRL7. The analysis has outlined that the security issues, at the state-of-the-art, depend mainly on the deployment of the solution in a development environment (debugging options activated and missing input validation).

5.3. Security assessment of the SPECS application

In order to offer an additional evaluation of the security of the SPECS Platform, we made an additional preliminary security assessment, based on penetration testing: the main goal is to identify the main security threats to which our platform is currently exposed.

As a starting consideration, we must note that the SPECS Platform and all the core components are developed as web applications; thus we performed our analysis adopting the common *web security* methodologies. In particular, we relied on the OWASP [14] methodologies and tools. The target of our web security analysis is the web site http://apps.specs-project.eu, which hosts our demonstrative application.

It is worth noting that the SPECS Platform is, at the state of the art, simply validated in laboratory and must not be considered as ready for production. This preliminary assessment identifies the main security issues in order to identify the process and the actions needed to secure the solution. The public web site, through which we offer (publically, but for a limited time) the full platform as a public cloud, is an additional result. Thus the following security analysis is a reference that can help with a new deployment. Note that the results of the presented analysis are valid for this specific deployment, not stating the quality of the overall solution.

In particular, we adopted a double strategy in order to identify SPECS security holes:

- **Approach 1**: Expert based penetration. We involved a set of security experts (mainly from EMC), that targeted the http://apps.specs-project.eu web site. The EMC report in Appendix 5 shows the result of such analysis and examples of security holes in the web application.
- **Approach 2**: Systematic penetration. Starting from the analysis of the experts and in order to clearly identify the main security threats we made a systematic analysis of security threats in the application through the OWASP tool.

Figure 6 briefly summarizes the iterative process we adopted in order to secure the SPECS platform. We adopted the OWASP ZAP [15] tool to perform automated penetration tests against the SPECS Platform. The tool generates a detailed report that summarizes the main threats and risks measured against the target web site.

Thanks to the penetration test results, we identify and classify, according to the STRIDE threat model [16], the threats to which the web application is exposed to.

As the last step of the iterative process, we identify a set of possible solutions to limit the security threats. Then the process restarts, applying again the systematic penetration testing.

The reports generated by the penetration analysis are available on SPECS SVN (maintained reserved).

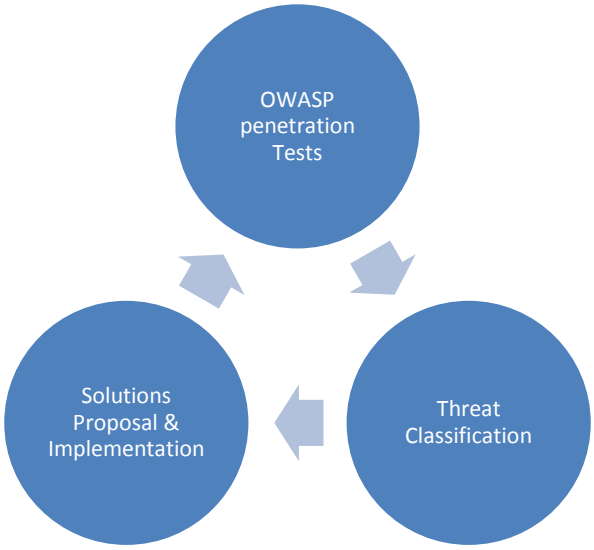


Figure 6. Security assessment analysis

Table 36 summarizes the threats identified on the target environment, according to the STRIDE classification and the risk level of the treats proposed (as measured by OWASP). Results refer to the last available iteration of the process.

Appendix 1 contains the list of analysed threats, reporting even their mapping against the WASC threat catalogue [17] and the Common Weakness Enumeration (CWE) [18].

STRIDE	LOW	MEDIUM	HIGH
Spoofing identity	T43		
Tampering with data		T39	T8
Repudiation			
Information disclosure	T38, T36, T41	T36	T9
Denial of service			
Elevation of privileges		T40	

Table 36. Existing threats and declared risk rating

Table 37 summarizes the threats identified per each module and component of the SPECS Platform.

Module	Component	Base Path	Threats ID
SLA Platform	SLA Manager	cloud-sla/slas/	T8
	Service Manager	cloud-sla/services-manager	T13
Negotiation	SLO Manager	sla-negotiation/	
Enforcement	Planning	sla-enforcement/sc-activities	T40
		sla-enforcement/supply-chains	
	Diagnosis	sla-enforcement/diagnosis	
		sla-enforcement/diag-activities	
	Planning	sla-enforcement/plan-activities	

	Implementation	sla-enforcement/plans	
		sla-enforcement/impl-activities	
	RDS	sla-enforcement/reconfigs	
		sla-enforcement/rem-activities	
		sla-enforcement/rem-plans	
Application	Platform interface	/platform-interface	T8, T9, T13, T14, T43
	Metric Catalogue	/metric-catalogue-app	
	Secure Web Container	/webcontainer-app-rev2/	
	Security Reasoner	/security-reasoner/	
Enabling Platform	Custom OS	:80/mos	T13, T36

Table 37. Security threats per component

5.4. Data regulation in SPECS

In the European Economic Area (EEA), the terms "Data protection rules" refer to the set of rules that protect personal data, where personal data is defined in as "*any information relating to an identified or identifiable natural person ('data subject')*". Personal data can include names, addresses, user and device identifiers (IP addresses and cookies), health and financial data, for example. These rules do not apply to other types of data such as intellectual property (e.g., music, movies) or business trade secrets and processes. Yet, almost all online services process personal data in some form or another. This applies to cloud services that are built with the SPECS platform, and we will therefore analyse the possible impact and advantages that SPECS offers for the processing of personal data.

Today the main text that defines data protection rules is[3] directive 95/46/EC [8]. Each member state has transposed this directive in national legislation, which often includes country-specific rules. At the end of 2015, the EU Parliament and the Council have reached an agreement on a regulation[4] that is destined to replace directive 95/46/EC with a more modern text. As opposed to Directive 95/46/EC that needed to be transposed in national law, the new text is a "regulation", which means that it will be directly applicable in 2018 when it likely enters into effect.

In addition to defining what "personal data" is, these rules also define the notion of a *controller* and a *processor*, where:

- A *controller* is the entity that defines the means and the purpose of a processing.
- A *processor* is the entity that acts on the instructions of a controller to implement some personal data processing.

Both the directive and the regulation establish some common principles, notably:

a) Personal data must be processed "fairly" and for a well-defined purpose.
b) Personal data must not be excessive for the purpose, and must not be kept longer than necessary.
c) Data subjects (users) must be informed about the processing of their personal data.

[3] There are also a few sectorial specific texts such as directive 2002/58/EC (Telecom sector).
[4] See http://europa.eu/rapid/press-release_IP-15-6321_en.htm

d) Data subjects (users) have the right to access their personal data and correct/delete it in some cases as well.
e) Personal data must be kept secure, with guarantees of confidentiality, integrity and availability.
f) Personal data must not be transferred to countries that do not offer a good level of protection.

The new regulation is expected to expand and add a few principles or requirements:

g) Companies processing personal data will be required to be "accountable": they will have to be able to demonstrate how they achieve compliance with the rules at any time.
h) For some types of "risky" data processing, a "data protection impact assessment" will be needed.
i) Personal data breaches (e.g., a hacker steals your data) will need to be notified to authorities and data subjects (users).
j) More responsibilities are put on the shoulders of "processors".

We will examine how SPECS can contribute to some of these principles.

5.4.1. Personal data in SPECS

To facilitate the description of personal data processing in a SPECS enabled offering, we will consider the following use case:

- FlowerPower is an SME of 50 employees, specialized in quick delivery of flowers and "get well" messages to hospitals, clinics and any health institution.
- The customers of FlowerPower use the website of FlowerPower to select flowers, add a message and provide delivery instructions.
- FlowerPower uses SPECS as a broker to select a cloud provider "CumuloNimbus", which will host its website and store customer data.
- FlowerPower selects some relevant SPECS security mechanisms to protect its data.

From a data protection perspective, FlowerPower is a *controller* while SPECS and CumuloNimbus are *processors*.

There are many sources of personal data in this use case:

1) The data of the customers of FlowerPower.
2) The data of the employees of FlowerPower (50 people).
3) The data of the employees of the SPECS broker.
4) The data of the employees of CumuloNimbus.

Here we will focus exclusively on the first case: customers of the data controller (FlowerPower) that uses the SPECS platform.

5.4.2. Benefits of SPECS in terms of information security

One of the key requirements of Directive 95/56/EC is security: the controller must "*implement appropriate technical and organizational measures to protect personal data against accidental or unlawful destruction or accidental loss, alteration, unauthorized disclosure or*

access, in particular where the processing involves the transmission of data over a network, and against all other unlawful forms of processing."

The SPECS platform supports this requirement by enabling the controller to select security mechanisms and negotiating the agreed security SLA. As an example, the Secure Storage application, which relies on DBB and E2EE mechanisms (see deliverables D5.2.1 and D5.2.2), offers a storage service, able to grant read-freshness and write-serializability. As shown in SPECS demos, the SPECS application notifies any unauthorized change of a document, generating a violation, and recovers the latest valid version of the document.

Deliverable D5.2.1 (table 2, page 24) illustrates the security controls that the Data Controller, as SPECS Application user, can request and agree on to demonstrate the adoption of appropriate technical and organizational measures to protect data.

5.4.3. Other benefits of the SPECS platform

Beyond information security, we examine further data protection related features of the SPECS platform.

5.4.3.1. Data location control

The ViPR and SPECS integration, proposed by EMC in deliverable D5.3, introduces the data geo-location security metric (ngDC_M2001, D5.3, page 21, Table 1). The innovative security SLA management of the ViPR controller, enabled by SPECS, enables the Data Controller (which acts as SPECS application customer) to request explicitly to the ViPR controller to acquire resources only in fixed availability zones.

5.4.3.2. Accountability

In the field of data protection, accountability describes *"the ability of parties to demonstrate that they took appropriate steps to ensure that data protection principles have been implemented"* [9]. It is a cornerstone of the future data protection regulation.

SPECS makes security objectives an explicit part of the signed SLA. Moreover, SPECS provides tools to monitor in real time the current level of security of the service that is used. This offers two clear benefits in terms of accountability for SPECS customers (such as FlowerPower in the use-case):

- SPECS customers can demonstrate that they designed their processing with explicit security properties in mind, as illustrated in the SLA.
- SPECS customers can report the current level of security of their cloud system, and take proactive measures if alerts are issued.

When dealing with authorities, these elements can help customers demonstrate that they conducting "due diligence" with respect to personal data processing, in acting in an accountable fashion.

6. Conclusions

By presenting all technical aspects of the integration of SPECS components and applications, and elaborating on performance, scalability, and security properties of the developed solutions, this deliverable concludes the report on SPECS integration activities.

In this document we have summarized the integration approach adopted in the project by reporting the integration plan and the continuous integration approach in SPECS as defined in deliverables D4.5.2 and D1.5.1. We have elaborated on integration examples by providing implementation details of integration scenarios for core components as well as the SPECS applications. Finally, we have defined the approach to performance and scalability analysis of the developed SPECS software, and presented the methodology adopted to evaluate security aspects of core components and applications. Since SPECS also processes personal data of cloud users, we have discussed how the data regulation is considered and adopted in SPECS.

Security review and assessment of the SPECS platform outlined that it is perfectly in line with the declared TRL levels (TRL3/4) and we identified the main issues to address in order to move the overall solution to higher TRL levels.

Performance analysis illustrates that the minimal configuration adopted is able to support up to one hundred users per minute, which implies the support of a cloud environment that delivers hundreds of VMs per second. Even if such performances are much more than needed for the purpose of the project, it is worth noticing that the framework can be scaled up simply by distributing components on multiple virtual machines, granting higher performances. The provided benchmarking solution, adopted for the performance analysis, can be used in order to fine tune such a solution according to the SPECS Owner's needs.

Last but not least, all components rely on MongoDB [21] for data persistence, which was configured to be scalable thanks to the activity related to monitoring (WP3, deliverable D3.4.2). Even if we never tested such solution (being out of the goal of the project), all components can be replicated and automatically synchronized, adopting ad-hoc configurations, which are common at the state of the art.

Bibliography

[1] "*Chef*", 2008. [Online]. Available online, https://www.chef.io/, last accessed in April 2016.

[2] "*Hosted SPECS Testbed web user interface*", 2015. [Online]. Available online, https://cloud.info.uvt.ro/, last accessed in April 2016.

[3] "*Atlassian Bamboo*", 2007. [Online]. Available online, https://www.atlassian.com/software/bamboo/, last accessed in April 2016.

[4] "*IeAT Bamboo – Build Dashboard*", 2015. [Online]. Available online, https://bamboo.services.ieat.ro/allPlans.action, last accessed in January 2016.

[5] "*Apache Maven*", 2004. [Online]. Available online, http://maven.apache.org/index.html, last accessed in April 2016.

[6] "Selenium", 2015. [Online]. Available online, http://www.seleniumhq.org/, last accessed in April 2016.

[7] "*SPECS Integration*", 2015. [Online]. Available online, https://bitbucket.org/account/user/specs-team/projects/SPint, last accessed in April 2016.

[8] "*Directive 95/46/EC of the European Parliament and of the Council of 24 October 1995 on the protection of individuals with regard to the processing of personal data and on the free movement of such data*", Official Journal of the European Union, L 281:0031-0050, 1995. Available online, http://eur-lex.europa.eu/legal-content/EN/TXT/?uri=uriserv:OJ.L_.1995.281.01.0031.01.ENG, last accessed in March 2016.

[9] Article 29 Data Protection Working Party, "Opinion 05/2012 on Cloud Computing", WP 196, 2012. Available online, http://ec.europa.eu/justice/data-protection/article-29/documentation/opinion-recommendation/files/2012/wp196_en.pdf, last accessed in March 2016.

[10] R. Jain, "*The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modelling*", Wiley-Interscience, New York, NY, April 1991.

[11] Gatling Corp, "*Gatling*", 2015. Available online, http://gatling.io/#/, last accessed in March 2016.

[12] The Open Web Application Security Project, "*Application Security Verification Standard (2014)*", 2014. [Online]. Available online, https://www.owasp.org/images/5/58/OWASP_ASVS_Version_2.pdf, last accessed in March 2016.

[13] "*Selenium*", 2015. Available online, http://www.seleniumhq.org/, last accessed in April 2016.

[14] "*OWASP*", 2016. Available online, https://www.owasp.org/index.php/Main_Page, last accessed in April 2016.

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

[16] Microsoft, "*STRIDE Threat Model*", 2002. Available online, https://msdn.microsoft.com/en-us/library/ee823878(v=cs.20).aspx, last accessed in April 2016.

[17] Web Application Security Consortium, "*WASC Project*", 2005. Available online, http://www.webappsec.org/, last accessed in April 2016.

[18] The MITRE Corporation, "*Common Weakness Enumeration*", 2015. Available online, https://cwe.mitre.org/, last accessed in April 2016.

[19] SPECS, "*SPECS Team Bitbucket account*", 2015. Available online, https://bitbucket.org/specs-team/, last accessed in April 2016.

[20] SPECS, "SPECS *Performance tests*", 2016. Available online, https://bitbucket.org/specs-team/specs-performance-benchmark, last accessed in April 2016.

[21] MongoDB, "*MongoDB*", 2016. Available online, https://www.mongodb.org/, last accessed in April 2016.

Appendix 1. SPECS prototypes

The following three tables provide a mapping of SPECS artifacts (core components, security mechanisms, and example applications) to the project's Bitbucket repositories and web sites, where the interested reader can find the SPECS prototypes.

SLA Platform	
SLA Manager	• https://bitbucket.org/specs-team/specs-core-sla_platform-sla_manager-api • https://bitbucket.org/specs-team/specs-core-sla_platform-sla_manager
Service Manager	• https://bitbucket.org/specs-team/specs-core-sla_platform-service_manager • https://bitbucket.org/specs-team/specs-core-sla_platform-service_manager-api
Interoperability Layer	• https://bitbucket.org/specs-team/specs-core-sla_platform-interoperability
Metric Catalogue	• https://bitbucket.org/specs-team/specs-core-sla_platform-security-metric-catalogue • https://bitbucket.org/specs-team/specs-core-sla_platform-security_metric_catalogue-api
Negotiation module	
SLO Manager	• https://bitbucket.org/specs-team/specs-core-negotiation-slomanager
Supply Chain Manager	• https://bitbucket.org/specs-team/specs-core-negotiation-supply_chain_manager
Security Reasoner	• https://bitbucket.org/specs-team/specs-core-negotiation-securityreasoner
Enforcement module	
Planning	• https://bitbucket.org/specs-team/specs-core-enforcement-planning
Implementation	• https://bitbucket.org/specs-team/specs-core-enforcement-implementation • https://bitbucket.org/specs-team/specs-core-enforcement-broker
Diagnosis	• https://bitbucket.org/specs-team/specs-core-enforcement-diagnosis
RDS	• https://bitbucket.org/specs-team/specs-core-enforcement-rds
Monitoring module	
Event Hub	• https://bitbucket.org/specs-team/specs-monitoring-eventhub
Event Archiver	• https://bitbucket.org/specs-team/specs-monitoring-event-archiver
MoniPoli Filter	• https://bitbucket.org/specs-team/specs-core-monitoring-monipoli
CTP	• https://bitbucket.org/specs-team/specs-monitoring-cloud-trust-protocol-adaptor • https://bitbucket.org/specs-team/specs-monitoring-cloud-trust-protocol-server
Nmap	• https://bitbucket.org/specs-team/specs-monitoring-nmap
Enabling Platform	
Launcher	• https://dashboard.cloud.specs-project.eu/#
Core Repository	• https://bitbucket.org/specs-team/specs-core-enforcement-repository
Mechanism Repository	• https://bitbucket.org/specs-team/specs-core-enabling_platform-repository
Vertical Layer	
User Management	• https://bitbucket.org/specs-team/specs-core-vertical_layer-user_manager
Auditing	• https://bitbucket.org/specs-team/specs-core-sla_platform-auditing
Security Tokens	• https://bitbucket.org/specs-team/specs-utility-security-tokens
Credential Service	• https://bitbucket.org/specs-team/specs-utility-credential_manager • https://bitbucket.org/specs-team/specs-enforcement-credentials-service (old version)

Security Mechanisms	
WebPool	• https://bitbucket.org/specs-team/specs-mechanism-enforcement-webpool
TLS	• https://bitbucket.org/specs-team/specs-core-enforcement-tls
SVA	• https://bitbucket.org/specs-team/specs-mechanism-enforcement-sva_dashboard • https://bitbucket.org/specs-team/specs-mechanism-monitoring-sva • https://bitbucket.org/specs-team/specs-mechanism-enforcement-sva_core • https://bitbucket.org/specs-team/specs-mechanism-enforcement-sva_vulnerability_manager • https://bitbucket.org/specs-team/specs-mechanism-monitoring-openvas
DoS	• https://bitbucket.org/specs-team/specs-mechanism-enforcement-dos • https://bitbucket.org/specs-team/specs-mechanism-monitoring-dos
E2EE & DBB	• https://bitbucket.org/specs-team/specs-enforcement-e2ee-koofr-client • https://bitbucket.org/specs-team/specs-mechanism-enforcement-e2ee-client • https://bitbucket.org/specs-team/specs-mechanism-enforcement-e2ee-server • https://bitbucket.org/specs-team/specs-mechanism-monitoring-e2ee-auditor • https://bitbucket.org/specs-team/specs-mechanism-monitoring-e2ee-adapter
AAA	• https://bitbucket.org/specs-team/specs-mechanism-enforcement-aaa • https://bitbucket.org/specs-team/specs-mechanism-enforcement-aaa-client

SPECS Applications	
Secure Web Container	• https://bitbucket.org/specs-team/specs-app-webcontainer-demo • https://bitbucket.org/specs-team/specs-app-webcontainer • https://bitbucket.org/specs-team/specs-app-webcontainer-rev2 • https://bitbucket.org/specs-team/specs-app-platform_interface
Metric Catalogue	• https://bitbucket.org/specs-team/specs-app-security_metric_catalogue
Security Reasoner	• https://bitbucket.org/specs-team/specs-app-securityreasoner

Appendix 2. SPECS artifacts in deliverables

The following table (part of it was initially reported in D1.1.3 at M24) presents the mapping among SPECS artifacts and the project's deliverables to enable the reader to locate the design and the implementation details for each SPECS artifact.

SPECS artifact		**Design**	**Implementation**
SLA Platform	SLA Manager	D1.4.1	D1.4.2
	Service Manager	D1.4.1	D1.4.2
	Interoperability Layer	D1.4.1	D1.4.2
	Metric Catalogue	D1.4.1	D1.4.2
Negotiation module	SLO Manager	D2.2.2	D2.3.2, D2.3.3
	Supply Chain Manager	D2.2.2	D2.3.2, D2.3.3
	Security Reasoner	D2.2.2	D2.3.2, D2.3.3
Enforcement module	Planning	D4.2.2	D4.3.2, D4.3.3
	Implementation	D4.2.2	D4.3.2, D4.3.3
	Diagnosis	D4.2.2	D4.3.2, D4.3.3
	RDS	D4.2.2	D4.3.2, D4.3.3
Monitoring module	Event Hub	D3.3	D3.4.1, D3.4.2
	Event Archiver	D3.3	D3.4.2
	MoniPoli Filter	D3.3	D3.4.2
	CTP	D3.3	D3.4.2
	Nmap	D3.3	D3.4.2
Enabling Platform	Launcher	D1.6.2	D1.6.1, D1.6.2
	Custom OS	D1.6.1, D1.6.2	D1.6.1, D1.6.2
	Core Repository	D1.6.1, D1.6.2	D1.6.1, D1.6.2
	Mechanism Repository	D1.6.1, D1.6.2	D1.6.1, D1.6.2
Vertical Layer	User Manager	D1.4.1	D1.4.2
	Auditing	D1.4.1	D1.4.2
	Security Tokens	D4.2.2	D4.4.2
	Credential Service	D4.2.2	D4.4.2
Security Mechanisms	WebPool	D4.2.2	D4.3.2
	TLS	D4.2.2	D4.3.2
	SVA	D4.2.2	D4.3.2
	DoS	D4.2.2	D4.3.3
	DBB	D4.2.2	D4.3.2
	E2EE	D4.2.2	D4.3.2
	AAA	D4.2.2	D4.3.3
Applications	Secure Web Container	D5.1.3	D5.1.3, D1.5.2
	Secure Storage	D5.2.1	D5.2.2
	ngDC	D5.3	D5.3
	AAAaaS	D5.4	D5.4
	Metric Catalogue	D5.1.3	D5.1.3, D1.5.2
	Security Reasoner	D2.3.1	D1.5.2
APIs		D1.3	D1.3

Appendix 3. Integration user guide

In the following we present a user guide to integration process in SPECS. We report the guidelines for preparing a Bitbucket repository for an integration test, setting up a Bamboo build plan, setting up a Bamboo deployment project, and finally running the integration test.

Preparing a Bitbucket repository

First, create a Bitbucket repository named *specs-integration-test-<scenario>*, where *<scenario>* is the name of the integration scenario or the name of the integration scenario family. For example, for the integration scenario family *Core-AB*, create a Bitbucket repository with a name *specs-integration-test-coreAB*.

Each integration scenario can have its own Bitbucket repository, but preferably similar integration scenarios implemented with similar technologies are joined in the same Bitbucket repository. Put the repository in the Bamboo project *SPECSintegration*.

For each integration scenario implement one or more tests using any testing framework which can be run with maven (Junit, TestNG). For easier debugging and identifying problems, it is useful that tests print as much as possible diagnostics information to the standard output which will be seen in the Bamboo logs.

The IP of the VM with integration testing environment running the SPECS platform is passed to tests through environment variable named SPECS_PLATFORM_IP. The value can be retrieved using the Java method *System.getenv()*:

```
System.getenv("SPECS_PLATFORM_IP")
```

Setting up the Bamboo build plan

For each integration scenario create a Bamboo build plan in the project *Integration* with the name equal to the scenario name. The build plan should have a job with following two tasks:

1. Source Code Checkout, which retrieves the corresponding repository from the Bamboo.
2. Maven 3.x, which runs the corresponding integration test(s).

In case the repository contains only one integration scenario, the Maven task's goal is:

```
clean test
```

In case the repository contains tests for more than one integration scenario, the Maven task's goal should specify which test(s) to run:

```
clean test -Dtest=<test class name>
```

For example, for the core integration scenario *Core-AB1*, the Maven taks's goal is:

```
clean test -Dtest=IntegrationScenarioCoreAB1Test
```

The IP of the integration VM running the SPECS platform can be passed to the integration test using an environment variable. Add the following to the field *Environment variables* in the Maven task configuration:

```
-DSPECS_PLATFORM_IP=${bamboo.integrationEnvironmentIp}
```

Here the *bamboo.integrationEnvironmentIp* is a Bamboo global variable containing the IP of the integration VM.

Setting up the Bamboo deployment project

SPECS components are installed by the Chef[5] client running on a VM using Chef recipes. Chef client runs periodically and checks if all recipes from the run list are installed. We use this behaviour of the Chef client to update SPECS components on the integration VM. After a Bamboo build plan has successfully completed, the deployment is triggered to undeploy the old version of the corresponding SPECS component. The Chef client detects that the components is missing and reinstalls the latest version.

To configure a Bamboo deployment project, take the following steps:

1. Create a deployment project and link it to the corresponding build plan.
2. Add an environment to the deployment project named, for example, *Integration*.
3. For this environment, define the task *Undeploy Tomcat Application* with the following settings:
    a. Tomcat Manager *URL: http://${integrationEnvironmentIp}:8080/manager*
    b. Tomcat Manager username and password corresponding to the user defined in the Tomcat configuration file *tomcat-users.xml*.
    c. Application context to which the corresponding SPECS component is deployed.
4. Add the trigger *After successful build plan* to the environment *Integration*.

Running integration tests

Integration tests can be run manually by running the corresponding Bamboo build plan or automatically according to a schedule. The output of the integration test can be checked in the Bamboo build plan logs.

[5] For the details about how Chef is used in SPECS, see deliverable D4.2.2.

Appendix 4. Performance analysis of the SLA Platform and the default SPECS application

We prepared *user profiles* for the following set of SLA Platform core components:

- SLA Manager
- Service Manager
- Metric Catalogue
- Interoperability Layer

The four tables below summarize the user profiles used to stress test each of the component. Note that, in the case of the Interoperability Layer, the tests only stress its interface to create new Virtual interfaces. In order to test it correctly, it should be stressed using the profiles of the other components, once it is configured in order to redirect their invocations.

Note that all SLA Platform components have a very simple behaviour and they are just a thin layer over their persistence solution (MongoDB): their relevance is in the metadata that they maintain (SLA documents with their state, Services metadata, metrics description) more than on the simple REST API offered.

User profile	Description	Scripts
SLA Create	Create a new SLA in the SLA Manager	• `SLAPostInject.scala` • `SLAPostRamp.scala`
SLA Get	Retrieve the list of SLA documents, get one of them and store it	• `SLAGetsInject.scala` • `SLAGetsRamp.scala`
SLA Sign	Retrieve the list of SLA documents, sign one of them	• `SLASignInject.scala` • `SLASignRamp.scala`

Table 38. SLA Manager user profiles for performance tests

User profile	Description	Scripts
Service Create	Create a new Mechanism in the Service Manager	• `ServicePostInject.scala` • `ServicePostRamp.scala`
Service Get	Retrieve the list of Mechanisms, get one of them and store it	• `ServiceRequestsInject.scala` • `ServiceRequestsRamp.scala`

Table 39. Service Manager user profiles for performance tests

User profile	Description	Scripts
Metric Create	Create a new metric in the catalogue	• `MetricPostInject.scala` • `MetricPostRamp.scala`
Metric Get	Retrieve the list of metrics, get one of them and store it	• `MetricRequestsInject.scala` • `MetricRequestsRamp.scala`

Table 40. Metric Catalogue user profiles for performance tests

User profile	Description	Scripts
Virtual Interface Create	Create a new virtual Interface in the Interoperability Layer	• `InteroperabilityPostInject.scala` • `InteroperabilityPostRamp.scala`
Virtual Interface Get	Retrieve the list of Virtual Interfaces, get one of them and store it	• `InteroperabilityGetsInject.scala` • `InteroperabilityGetsRamp.scala`

Table 41. Interoperability Layer user profiles for performance tests

The default SPECS application user profiles were built according to the proposed user interface and to the SLA life cycle phases. Table 42 summarizes the proposed user profiles. Note that Implementation scripts cannot be adopted against the real integrated environment, due to the limited availability of resources on the testing environment. We can only stress the Implementation component by disabling the real brokering function. According to such considerations, we made a single user profile, associated to the Wizard, which orchestrates all requests to the platform, so summarizing the overall behaviour during the real application execution. Performance figures reported in the tables present all calls invoked by the application wizard.

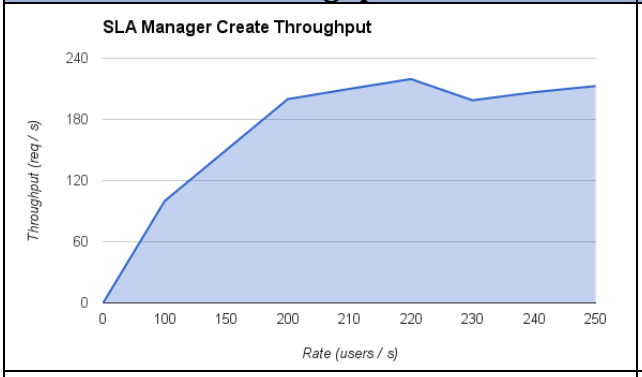
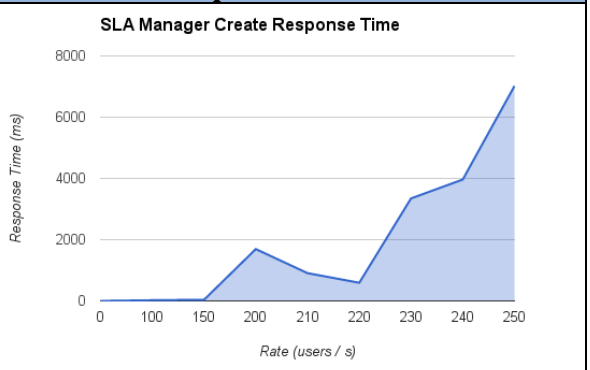
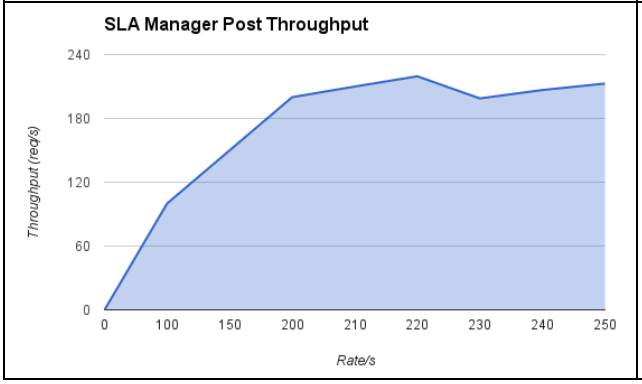
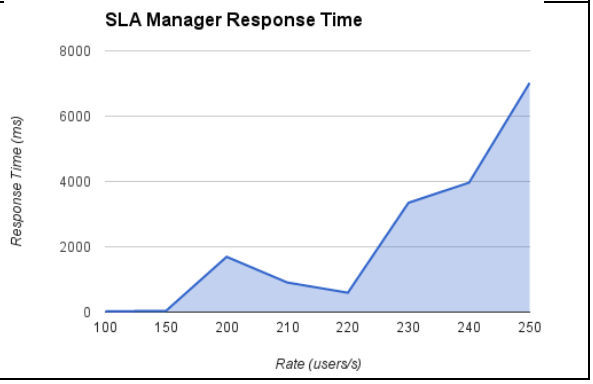
User Profile	Description	Scripts
Wizard	Perform the full negotiation process, selecting all the proposed capabilities	• `WizardInject.scala` • `WizardRamp.scala`

Table 42. SPECS application user profiles for performance tests

In the following we present results of performance tests for the components of the SLA Platform and the default SPECS application.

SLA Manager

In the table below we present performance results for the SLA Manager component. In the left column we illustrate the *throughput* granted by the SLA Manager under an increasing load up to 250 users per second, and in the right column we present the *response time* of the associated tests. Throughput is much lower when operations require the read operations (that implies a listing of all SLAs and access to one of them), allowing only ten or twelve of concurrent users per second (i.e., 120 users per minute).

Throughput	Response Time
	
	

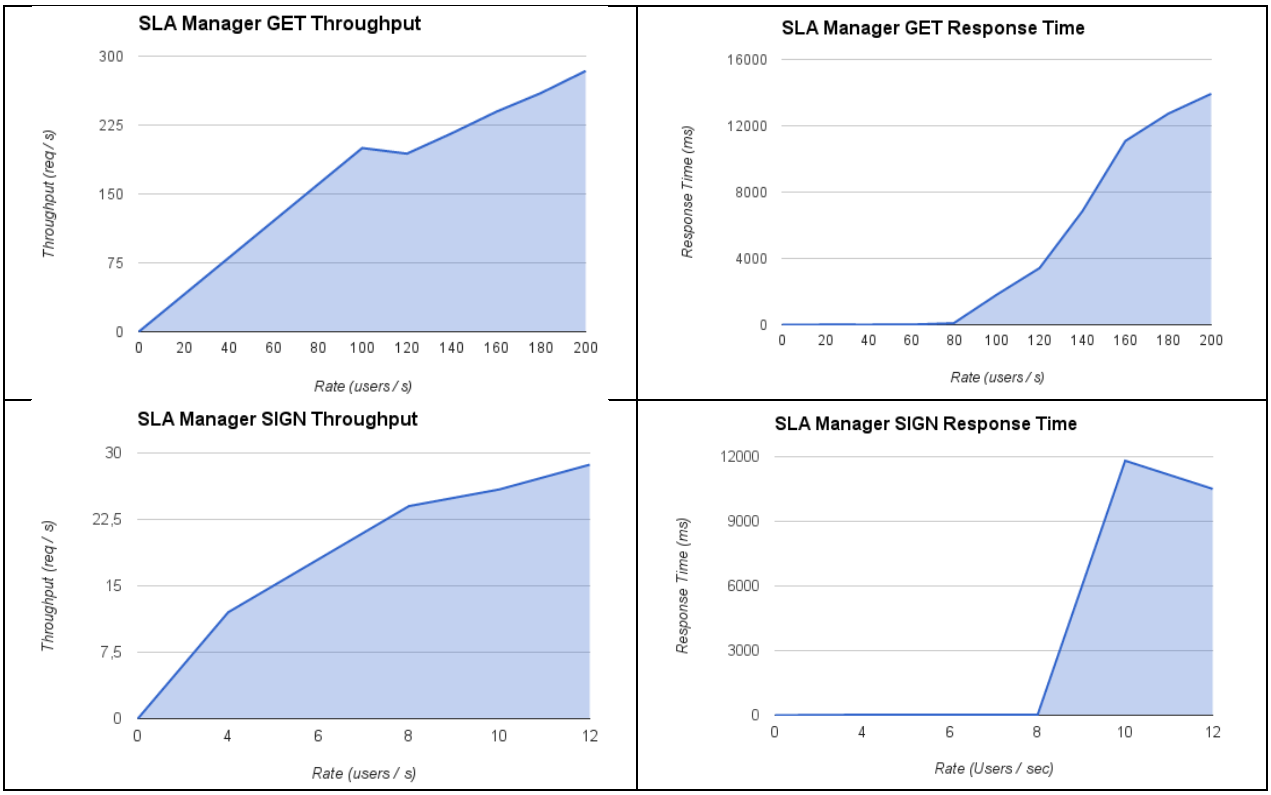
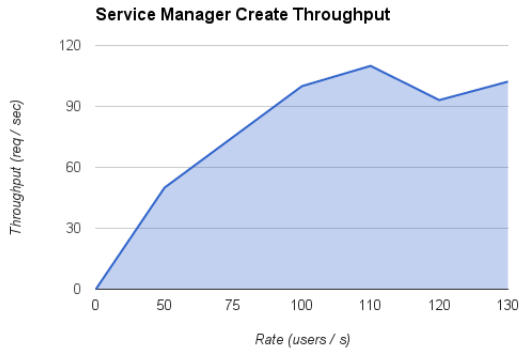
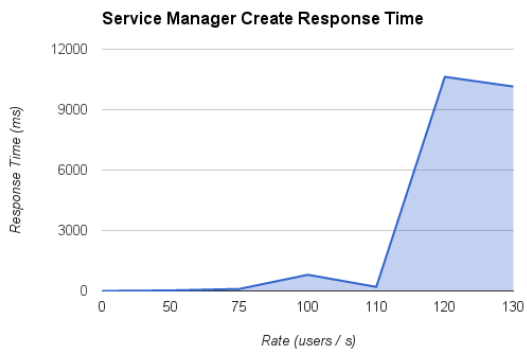


Table 43. Performance results for the SLA Manager

Service Manager

In the table below we present performance results for the Service Manager component. In the left column we illustrate the *throughput* granted by the Service Manager under an increasing load up to 130 users per second, and in the right column we present the *response time* of the associated tests.

Throughput	Response Time
	

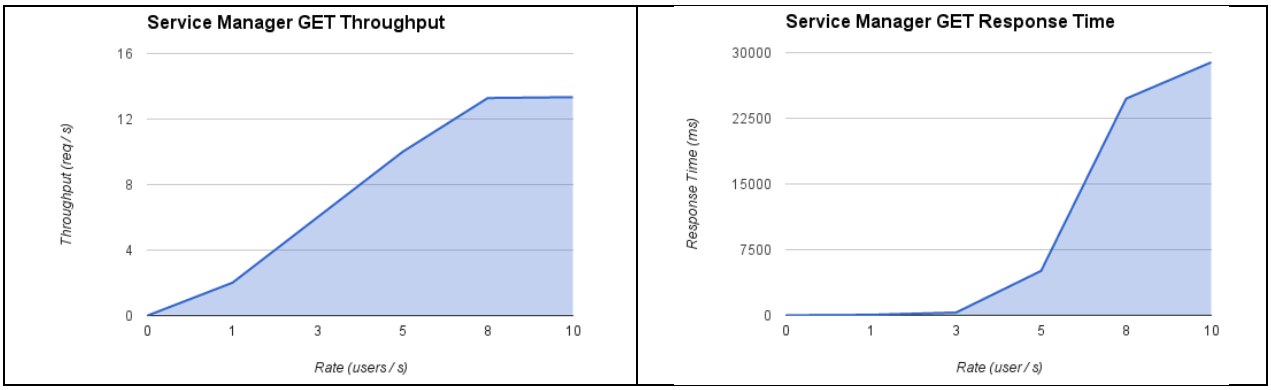


Table 44. Performance results for the Service Manager

Metric Catalogue

In the table below we present performance results for the Metric Catalogue component. In the left column we illustrate the *throughput* granted by the Metric Catalogue under an increasing load up to 450 users per second, and in the right column we present the *response time* of the associated tests.

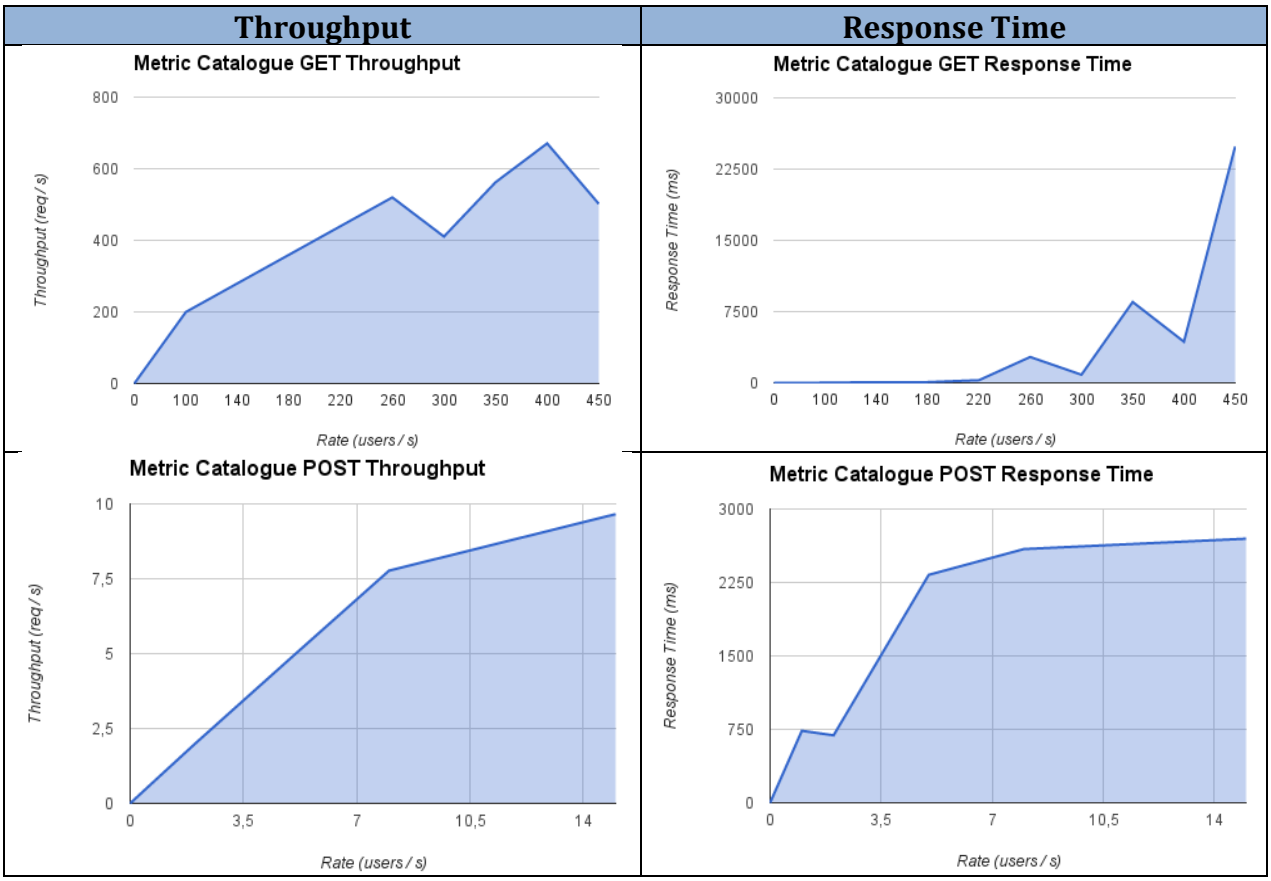


Table 45. Performance results for the Metric Catalogue

Interoperability Layer

In the table below we present performance results for the Interoperability Layer component. In the left column we illustrate the *throughput* granted by the Interoperability Layer, and in the right column we present the *response time* of the associated tests. Note that the tests

illustrate the number of new virtual interfaces created (an operation that is performed rarely).

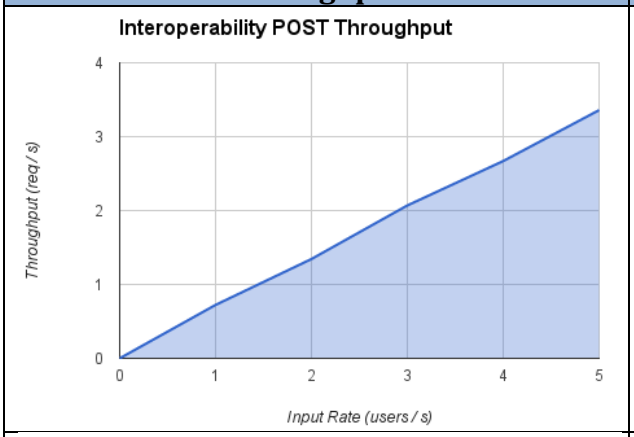
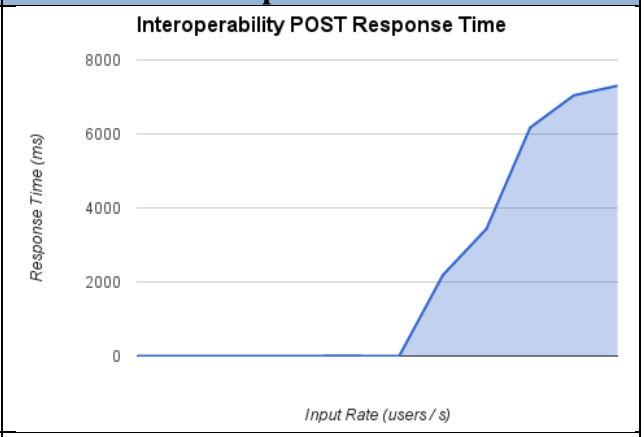
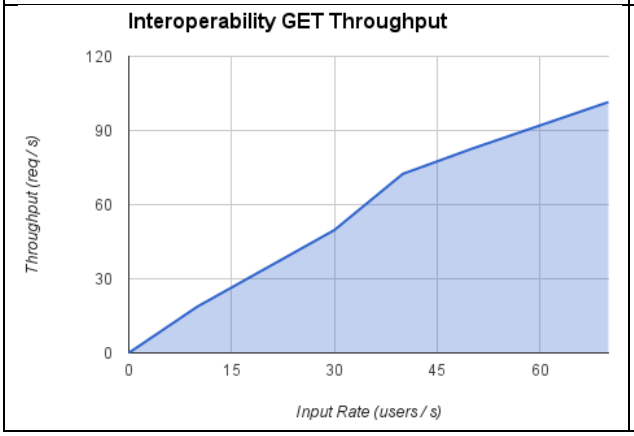
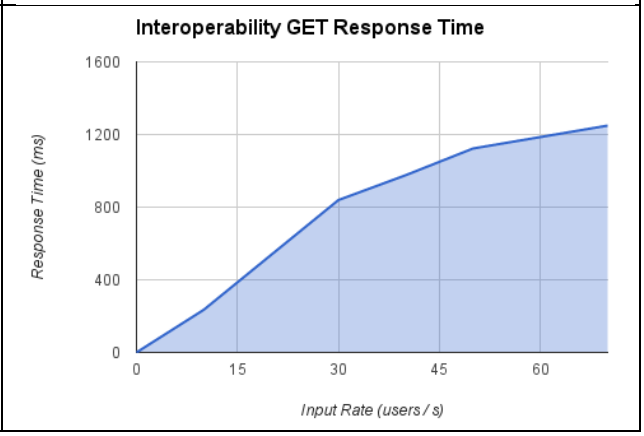
Throughput	Response Time
	
	

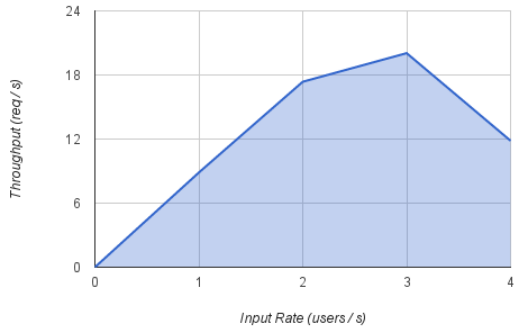
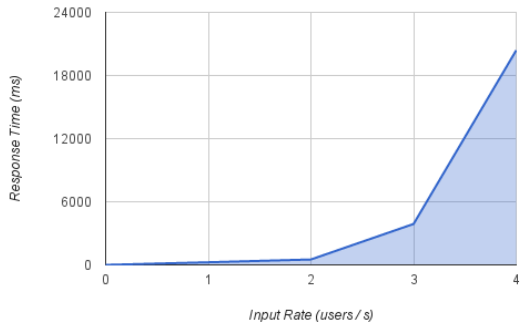
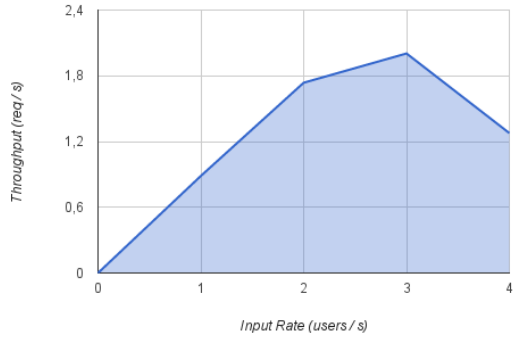
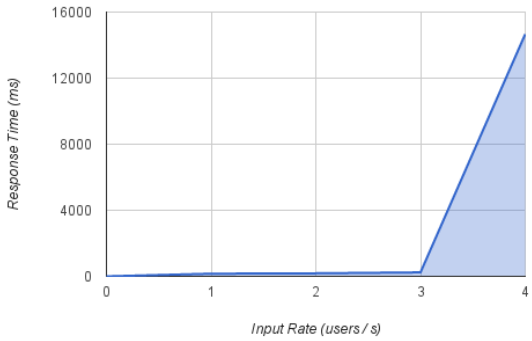
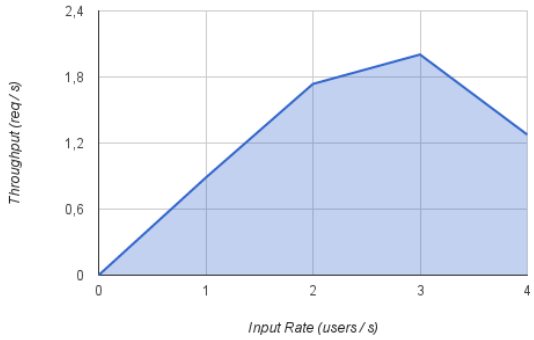
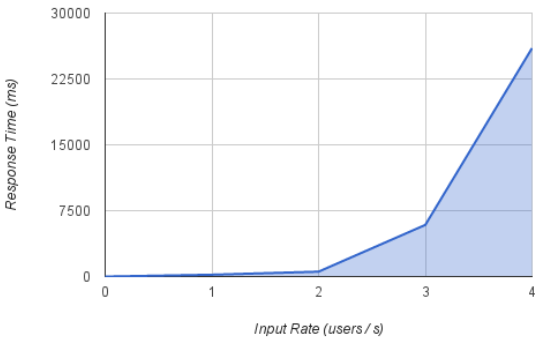
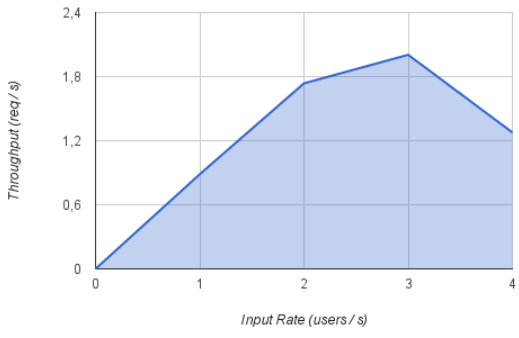
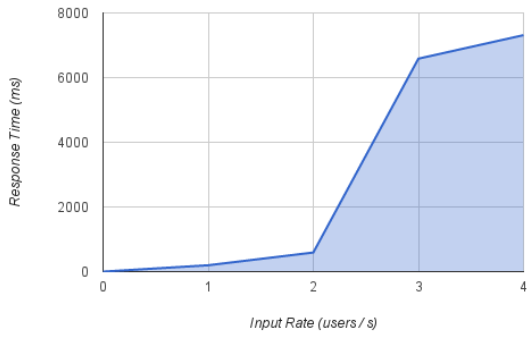
Table 46. Performance results for the Interoperability Layer

Default SPECS application

In the table below we present performance results for the default SPECS application. In the left column we illustrate the *throughput* granted by the SPECS application, and in the right column we present the *response time* of the associated tests. In the graphs below, we report the values for each action of the application wizard. The application wizard implies the execution of all the services offered by the SPECS Platform, according to the flow discussed in deliverable D1.3.

It is worth noticing that the application can manage about 3-4 user per second (i.e. about 120 user per minute). The wizard results in the acquisition of a varying number of VMs between 3 and 5, as a consequence 100 user per minute, implies about 300 VMs delivered per minute (SPECS testbed can deliver at most 30 VMs in total).

According to such result, even the minimal configuration proposed (all components hosted in the same VM, excluded only the Chef server) with only 1 GB of RAM memory enables to manage a little datacenter (few hundreds of VMs available).

Throughput	Response Time
SPECS App Wizard - Throughput	SPECS App Wizard - Response Time
	
SPECS App Welcome - Throughput	SPECS App Welcome - Response Time (ms)
	
SPECS App GET Capabilities - Throughput	SPECS App GET Capabilities - Response Time
	
SPECS App GET Controls - Throughput	SPECS App GET Controls - Response Time
	

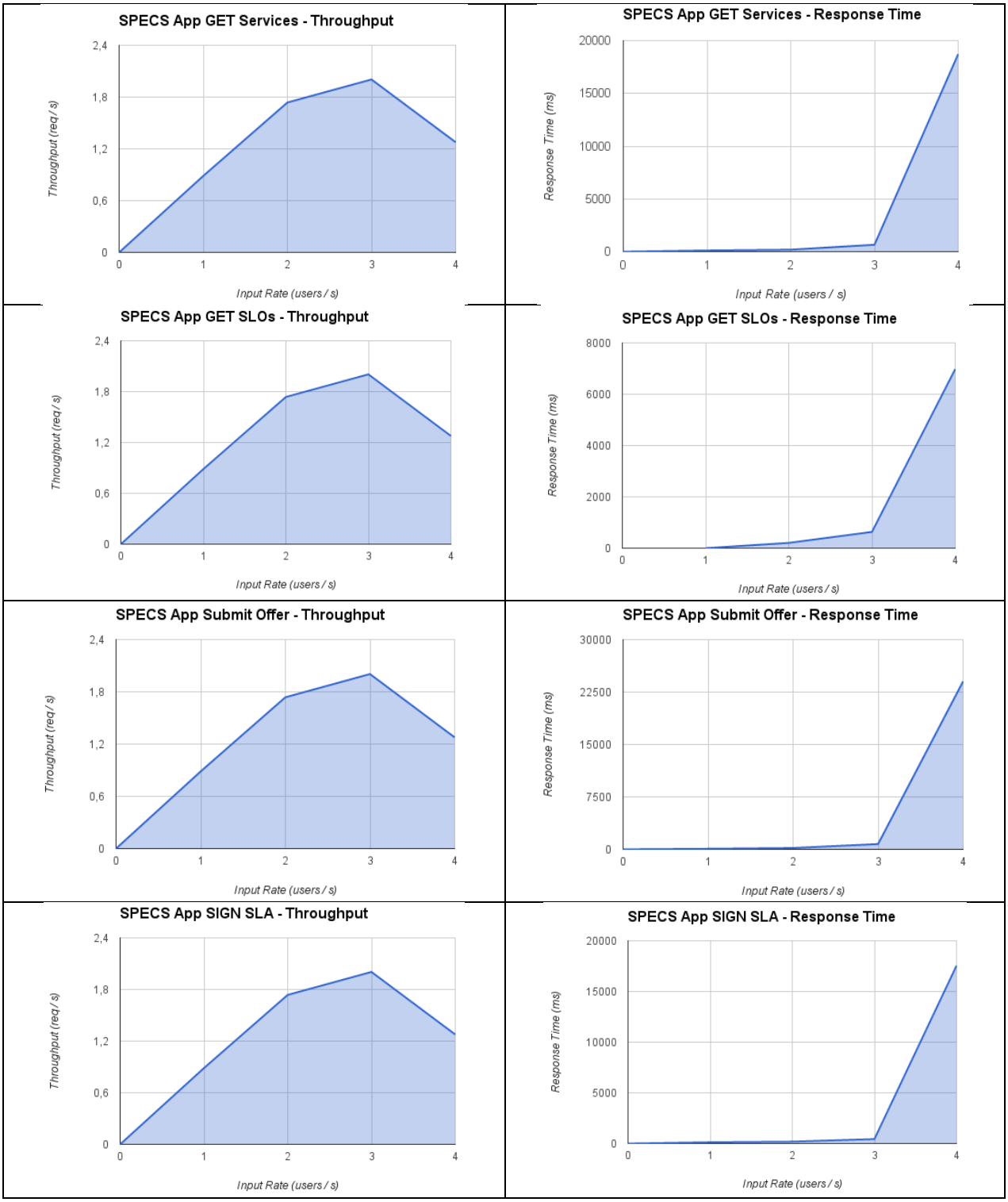


Table 47. Performance results for the default SPECS application

Appendix 5. SPECS application penetration testing

In the following we present results of the penetration testing of the default SPECS application.

Cross Site Scripting

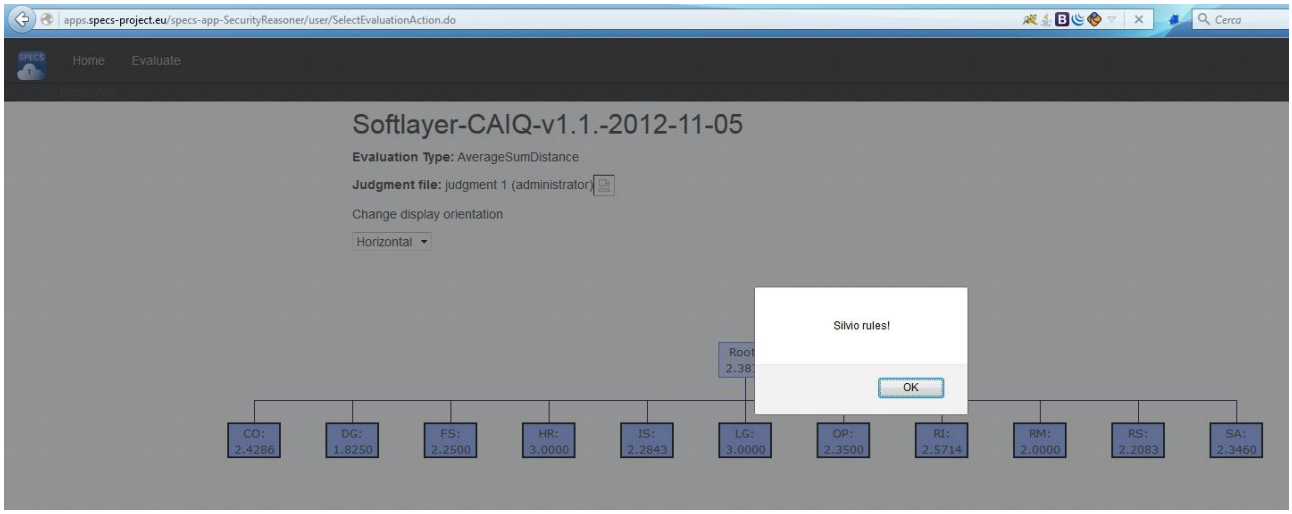
The application is affected by several Cross Site Scripting vulnerabilities. The issue has been verified using simple javascript codes (iframe, alert popup).

Below are the URLs and the screenshots of the affected pages.

Example 1: Page: */specs-app-SecurityReasoner/user/SelectEvaluationAction.do*

Request example:

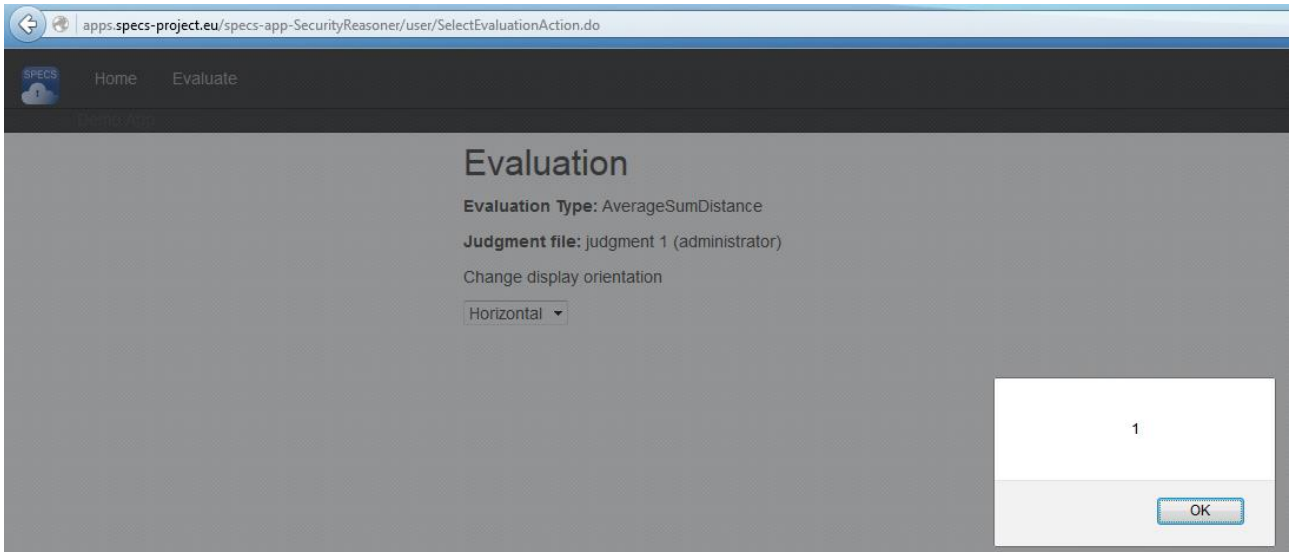
```
selectedCaiqs=78%24Softlayer-CAIQ-v1.1.-2012-11-
05&selectedJudgments=27%24judgment+1+%28administrator<iframe
onload=alert("Silvio rules"></iframe>%29&metodo=singleEvaluation
```



Example 2: Page: */specs-app-SecurityReasoner/user/SelectEvaluationAction.do*

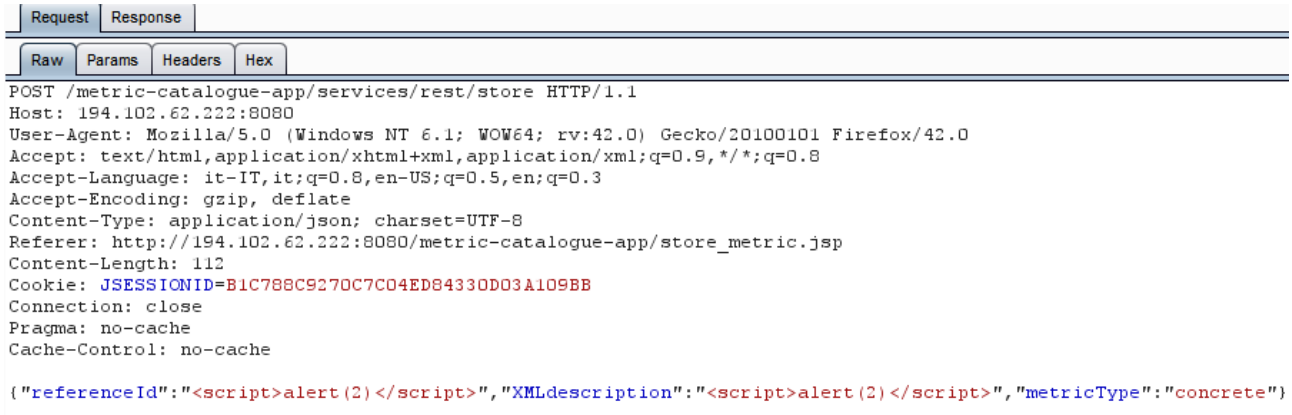
Request example:

```
selectedCaiqs=76%24Amazon+EC2+-CAIQ-v2-2013-11-
01</script><script>alert(1)</script>&selectedJudgments=27%24judgment+1+%
28administrator%29&metodo=singleEvaluation
```

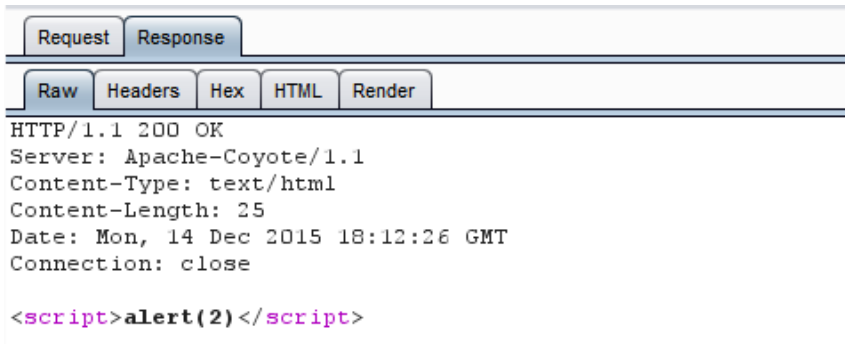


Example 3: Page: */metric-catalogue-app/services/rest/store*

Request example:



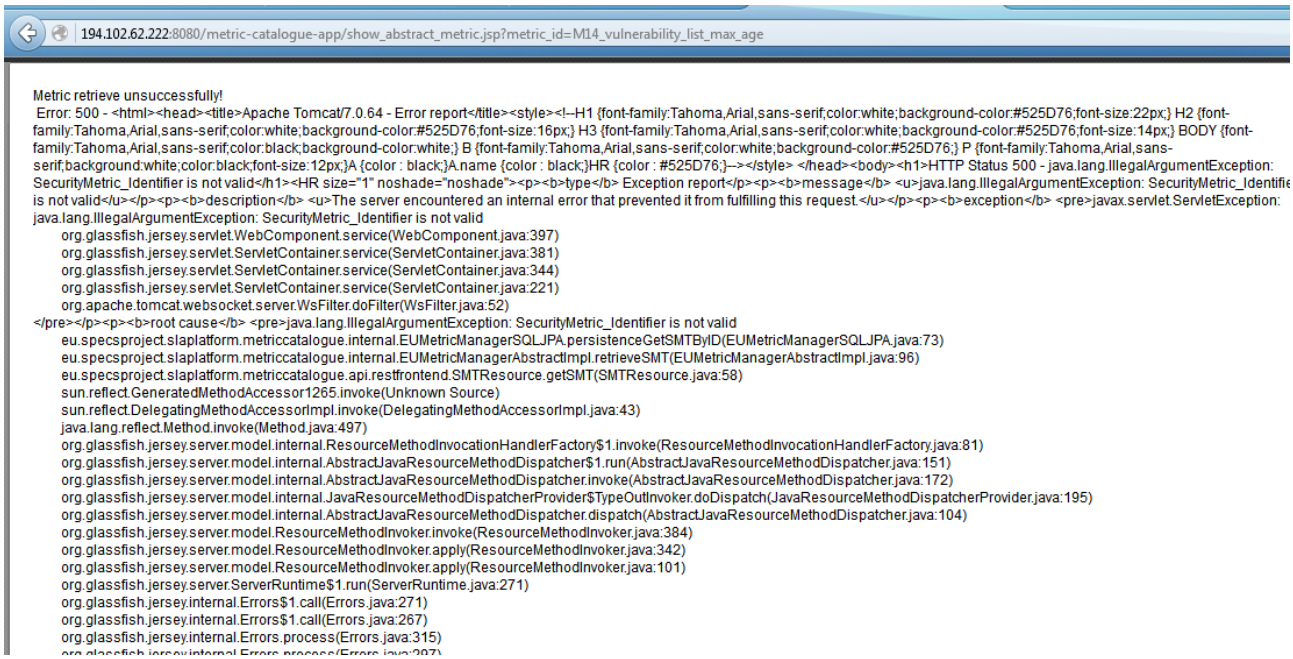
Output:



Solution
It is necessary to implement an input validation and sanitization mechanism on all the pages of the application to prevent the injection of malicious code. Output encoding is necessary, too.

It is not recommended to create custom filters; anyway, the validation should not be based on the concept of "blacklist", specifying which characters should be removed, but on the "whitelist", allowing only the characters that are expected.

Applicative errors not handled correctly
The application does not handle applicative errors, showing details which contain sensitive information about the source code and the version of the web application server, as shown in the following screenshot below.



It's easy to generate the output, simply navigating the application.

Solution
The application should handle all errors with custom pages to prevent disclosure of information (stack trace) that could be used by an attacker to perform a more focused attack against the app.

Minor problems
In the following we report minor problems that have been detected for the default SPECS application:

- The login page for manage tomcat exposed on http public (dictionary attack open).
- Monitor SLA does not work (at the time of the check).
- http://apps.specs-project.eu/examples/jsp/snp/snoop.jsp should not be exposed (it gives information to attackers).
- http://apps.specs-project.eu/examples/servlets/index.html should not be present (it could be used to inject code inside the server).

- http://apps.specs-project.eu/examples/jsp/ should not be present (it could be used to inject code inside the server).
- http://apps.specs-project.eu/examples/jsp/source.jsp error not managed.
- http://apps.specs-project.eu/docs/ should not be exposed (exposes tomcat version).
- http://apps.specs-project.eu/specs-app-webcontainer-demo/services/rest/slaTemplate should be double checked on Firefox; the negotiation process does not work.

In the following we report minor problems that have been detected for the Metric Catalogue application:

- The “Get Metric Database” returns an empty file.
- The “restore metric backup” returns 404 error.
- Specifying string as number for a metric ID returns 404 error code.
- http://apps.specs-project.eu/metric-catalogue-app/services/rest/retrieve/ output encoding managed properly injecting code.

Appendix 6. Threat catalogue

In the following, we present the list of threats (each with an ID, name, and a description) analysed during the security assessment of the default SPECS application. The list is based on the following sources:

- SRC1: CSA Survey on top threats to cloud computing in 2015[6]
- SRC2: OWASP Top 10 Project 2013[7]
- SRC3: Toward a Threat Model for Storage Systems[8]
- SRC4: OWASP ZAP[9]

We categorise each threat using the Microsoft's STRIDE threat model[10]:

- S = Spoofing
- T = Tampering with data
- R = Repudiation
- I = Information disclosure
- D = Denial of service
- E = Elevation of privileges

Additionally we map specified threats against the WASC catalogue[11] and the Common Weakness Enumeration (CWE)[12].

---

[6] CSA, "*Survey for the CSA Top Threats to Cloud Computing 2015*", 2015. Available online, https://cloudsecurityalliance.org/media/news/survey-for-the-csa-top-threats-to-cloud-computing-2015-report-is-open/, last accessed in April 2016.
[7] OWASP, "*OWASP Top 10 2013*", 2013. Available online, https://www.owasp.org/index.php/Top_10_2013-Top_10, last accessed in April 2016.
[8] R. Hasan, S. Myagamar, A. J. Lee, W. Yurcik, "*Toward a Threat Model for Storage Systems*", in Proceedings of the StorageSS'05, the 2005 ACM Workshop on Storage Security and Survivability, pp. 94-102, 2005. Available online, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.60.3820&rep=rep1&type=pdf, last accessed in April 2016.
[9] "*OWASP Zed Attack Proxy Project*", 2016. Available online, https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project, last accessed in April 2016.
[10] Microsoft, "*STRIDE Threat Model*", 2002. Available online, https://msdn.microsoft.com/en-us/library/ee823878(v=cs.20).aspx, last accessed in April 2016.
[11] Web Application Security Consortium, "*WASC Project*", 2005. Available online, http://www.webappsec.org/, last accessed in April 2016.
[12] The MITRE Corporation, "Common Weakness Enumeration", 2015. Available online, https://cwe.mitre.org/, last accessed in April 2016.

ID	NAME	Description	STRIDE	Source	WASC ID	CWE ID
T1	Account Hijacking	In account hijacking, a hacker uses a compromised account to impersonate the account owner. Typically, account hijacking is carried out through social engineering, phishing, sending spoofed emails to the user, password guessing or a number of other hacking tactics. In many cases, the outcome of an account hijacking is the hacker will have full system access and the ability to laterally access other systems on the target user network. The effective breach scope may expand to other services, such as financial and social networks, due to password re-use across services.	S	SRC1	WASC-18	
T2	Advanced Persistent Threats (APTs)	An advanced persistent threat (APT) is a system attack in which an unauthorized actor gains access to the infrastructure and remains undetected. The intention of an APT attack is to locate and steal data and evade detection, rather than to cause damage to the network or organization. APT attacks target organizations in sectors with high-value information, such as national defence, manufacturing, infrastructure, medical, scientific, and the financial industry.	R	SRC1		
T3	Broken Authentication and Session Management	The application procedures related to authentication and session management are often implemented incorrectly, allowing attackers to compromise passwords, keys, session tokens, or exploit weaknesses implementative to impersonate other users.	S	SRC2	WASC-01 WASC-11 WASC-12 WASC-18 WASC-37 WASC-47	CWE-306 CWE-287 CWE-307 CWE-345 CWE-798 CWE-330 CWE-384 CWE-613
T4	Cross-Site Request Forgery (CSRF)	A CSRF attack forces the victim's browser to send an HTTP request properly forged, including the victim's session cookie and any other authentication information, to a vulnerable web application. This allows the attacker to force the victim's browser to generate requests that the vulnerable application will believe legitimately sent by the victim.	T	SRC2	WASC-09	CWE-352
T5	Cross-Site Scripting (XSS)	XSS flaws occur when a web application receives data from unreliable sources, and send them to a browser without proper validation and / or	T	SRC2	WASC-08 WASC-24	CWE-79 CWE-93

		"escaping". The XSS allows attackers to execute malicious scripts on the browser of the victims; these scripts can hijack the user's session, deface the website or redirect the user to a malicious site				
T6	Data Breaches	A data breach is an incident in which sensitive, protected or confidential data has potentially been viewed, stolen or used by an individual unauthorised lo do so. Data breaches may involve personal health information (PHI), personally identifiable information (PII), trade secrets or intellectual property.	I	SRC1		
T7	Denial of Service	DoS and DDoS are both denial-of-service attacks. The attacks work by requesting more resources from a server than the server has available. In the case of DoS, it is an attack that originates from a single device, as opposed to DDoS which is distributed and relies on multiple devices.	D	SRCS1	WASC-10	CWE-67 CWE-134 CWE-285 CWE-364 CWE-382 CWE-400 CWE-412 CWE-479 CWE-512 CWE-524 CWE-589 CWE-594 CWE-606 CWE-617 CWE-646 CWE-730 CWE-772 CWE-775 CWE-776 CWE-781 CWE-799 CWE-824 CWE-825

						CWE-826 CWE-828 CWE-831 CWE-862 CWE-863 CWE-922 CWE-927 CWE-941
T8	Injection	The Injection Flaws, such as SQL Injection, OS Injection and LDAP injection, occur when data not validated are sent as part of a command or query to their interpreter. The data can deceive the interpreter running commands not provided or accessing data for which you have no authorization.	T	SRC2	WASC-05 WASC-19 WASC-20 WASC-23 WASC-25 WASC-28 WASC-29 WASC-30 WASC-31 WASC-36 WASC-39 WASC-46	CWE-98 CWE-426 CWE-73 CWE-89 CWE-564 CWE-20 CWE-91 CWE-113 CWE-158 CWE-90 CWE-88 CWE-78 CWE-97 CWE-643 CWE-652
T9	Insecure Direct Object References	When a developer exposes a reference to an internal implementation object, such as a file, directory, or a key in a database, it has a direct reference to an object. Without proper access control or other protection, attackers can manipulate these references to access unauthorized data.	I	SRC2	WASC-01 WASC-02 WASC-33	CWE-434 CWE-287 CWE-862 CWE-863 CWE-22
T10	Man in the Middle attack	MITM is an attack where the attacker secretly relays and possibly alters the communication between two parties who believe they are directly communicating with each other.	S	SRC1	WASC-32	

T11	Missing Function Level Access Control	Many applications check the level of access rights before its functionality is made visible in the user interface. However, applications need to perform access control on the server each time the feature is accessed. If access requests are not verified, the attackers can falsify them to gain unauthorized access to features.	E	SRC2	WASC-02 WASC-21 WASC-34	CWE-285 CWE-799 CWE-084 CWE-425
T12	Over-privileged application and accounts	Threat aimed to gain privileged access to resources for gaining unauthorized access to information or to compromise a system.	E	SRC1		
T13	Sensitive Data Exposure	Many web applications do not adequately protect data such as credit card numbers or authentication credentials. Attackers can take possession of the data or take advantage of the weaknesses in the security measures for the theft of credentials, for fraudulent transactions with CdC, etc. This type of data, require additional protective measures, such as encryption for data in transit, as well as special precautions when they are exchanged with the browser.	S	SRC2	WASC-50 WASC-04	CWE-311 CWE-327 CWE-759 CWE-326
T14	Unauthorized access to admin interface	Threat aimed to gain privileged access to resources for gaining unauthorized access to information or to compromise a system.	E	SRC1	WASC-15 WASC-17 WASC-14	
T15	Invalidated Redirects and Forwards	Web applications often redirect or forward users to other pages or sites and use data not validated to determine the destination pages. Without proper validation, attackers can redirect victims to phishing or malware sites, or use forwards to access unauthorized pages.	T	SRC2	WASC-38	CWE-601
T16	Weak Identity, Credential & Access Management	Lack of highly scalable identity access management systems, lack of multi-factor authentication capabilities, weak password usage, and lack of ongoing automated rotation of cryptographic keys, passwords, and certificates. Furthermore, hygiene of credentials ranging from embedding in source code and distribution in publicly available source code may be considerations.	I	SRC1		
T17	Sniffing Storage Traffic	Storage traffic on dedicated storage networks or shared networks can be sniffed revealing data, metadata, and storage protocol signalling.	T	SRC3		
T18	Snooping on Buffer Cache	Most file systems utilize buffer caches to read and write storage blocks from and into the storage media. This is the norm regardless of the file system	T	SRC3		

		technology used. The buffer caches are allocated on demand. If an attacker can snoop into the buffer caches in memory she can access storage blocks and hence stored information she is not authorized to access.				
T19	Snooping on Deleted Storage Blocks	In most file systems, storage blocks are allocated to files on demand. When a file is deleted, the storage block contents are not necessarily erased. Rather, most of the storage systems implement file deletion by erasing the file name and links from metadata and deleting the file i-node. Thus, data contents can be left un-erased in deleted and now free storage blocks. By accessing these storage blocks, it is possible for an attacker to gain access to sensitive data.	T	SRC3		
T20	Snooping on Deallocated Memory	Although most modern software deallocate data in memory after its last usage, it is possible for attackers to snoop on deallocated memory because the content of freed memory stays intact until it gets overwritten. Chow et al. point out in that after deallocation, sensitive data such as passwords, social security numbers, and credit card numbers, often remain in memory indefinitely, possibly for days. This increases the risk of exposing sensitive data when a system is compromised, or of data being accidentally leaked due to unexpected feature interactions such as core dumps, logging, etc. One solution to this problem is to reduce data lifetime by zeroing at time of deallocation.	T	SRC3		
T21	File System Profiling	File system profiling attacks attempt to use access type, timestamps of last modification, file names, and other file system metadata to gain insight about storage system operation. For example, if a set of files are accessed in regular patterns, the attacker may infer the importance, function, and possibly even the content of these files.	T	SRC3		
T22	Modifying Metadata	Modifying metadata will disrupt a storage system. In any file system, if the i-node or file table are corrupted, the storage linked to the metadata cannot be accessed.	T	SRC3		
T23	Subversion Attacks	Attacks which modify operating system (OS) commands, kernel system calls, and/or storage system drivers to cause the wrong files, metadata or blocks to be modified or deleted.	T	SRC3		

T24	Exhausting Log, Data and Metadata Space	Storage systems use different types of logging. In log-structured file systems, the whole file system is a series of logs. An attacker can create a large number of small modifications to fill up the log space and lock up the system. Moreover, an attacker can create a large number of data files with random content to use up the available disk space. An attacker may create also many empty/small/hidden files. While each file uses only a small amount of metadata space, a large number of metadata entries will degrade storage system performance.	D	SRC3		
T25	Creating Redundant Versions	Some versioning file systems, like S4 and Elephant create multiple versions of objects. Taking advantage of this, an attacker may launch a DoS attack by creating multiple versions of objects with minimal changes that will eventually exhaust storage space.	D	SRC3		
T26	Exhausting File Handles	In most storage systems, file handles are used to access files, and these are locked until the file is closed. Also, file systems usually have a fixed number of file handles. An attacker may create a DoS by opening up multiple files but not closing them, thereby holding the file handle and degrading storage system performance.	D	SRC3		
T27	Deletion of Data	Deleting data or metadata is an extreme DoS attack but also one that is easily detectable and possibly recoverable given versioning or backups in time or space. If the deleted data is unrecoverable, the cost may range from insignificant to incalculable. Deleting system and network logs is commonly used by attackers to cover their attack traces.	D	SRC3		
T28	Storage Device Masquerading	An attack storage device authenticates as a legitimate storage device to the OS in order to access/modify/deny data or metadata.	S	SRC3		
T29	Flash Memory Attacks	Attacks on flash memory are designed to force inordinate numbers of erase cycles to exhaust that capability.	D	SRC3		
T30	Power Disruption	If the power supply to a storage device is disrupted, storage systems can become unavailable and data/metadata lost. Many storage systems have backup power sources for this reason; however, even in these cases long term power disruption is possible.		SRC3		

T31	Network Disruption	Regardless of the underlying network technology, any hardware component or cable disruption to the network between the user and the storage system can degrade or disable storage.		SRC3		
T32	Storage Theft	Thefts of storage media, storage devices, and computers containing storage systems occur. Recently, there has been an epidemic of thefts of unencrypted storage tapes containing confidential customer information. The decreasing size of portable storage combined with increasing capacity—e.g., a USB memory stick with 4GB capacity—makes it easier to steal storage media. Although such thefts require low levels of sophistication on the attacker's part, they may result in large economic and security damages unless the stolen data/metadata is encrypted and replicated.	T	SRC3		
T33	Data Recovery from Discarded Storage Media	Neglecting to properly sanitize storage media before disposing of them allows attackers (or other third parties) access to data and metadata. Proper sanitation techniques include:, e.g. overwriting, degaussing, and encryption (along with destruction of corresponding decryption key).		SRC3		
T34	Physical Destruction of Storage Media	Storage media can be physically destroyed by attackers using disintegration, incineration, pulverization, shredding, or melting. If storage media is intentionally destroyed by the owner with a purpose of retiring it, the data may still be recoverable. Hughes demonstrates that even after shooting at a hard disk with a bullet, it is still possible to read data using special instruments such as a Magnetic Force Microscope.		SRC3		
T35	Hardware Trojan	A USB driver can be exploited to load malicious software. Barrall et al. describe how a custom-built USB device can fool an operating system into believing the device is any form of USB peripheral. Attackers can load malicious software, such as a keystroke logger, onto a target system simply by physically plugging the device into a USB port, bypassing the built-in OS security. A file containing harvested passwords can be retrieved through the USB port after a few days or a week.		SRC3		
T36	Security Misconfiguration	Good security requires having a secure configuration defined and deployed for the application, frameworks, application server, web server, database server, and platform. Secure settings should be defined, implemented, and		SRC2	WASC-13 WASC-14 WASC-15	CWE-209 CWE-219 CWE-200

		maintained, as defaults are often insecure. Additionally, software should be kept up to date.			WASC-16 WASC-17	CWE-754 CWE-16 CWE-548 CWE-250 CWE-732 CWE-280 CWE-538 CWE-552
T37	Using Components with Known Vulnerabilities	Components, such as libraries, frameworks, and other software modules, almost always run with full privileges. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications using components with known vulnerabilities may undermine application defenses and enable a range of possible attacks and impacts.		SRC2		
T38	Overly Permissive Cross-domain Whitelist	The software uses a cross-domain policy file that includes domains that should not be trusted. The software uses a cross-domain policy file that includes domains that should not be trusted. A cross-domain policy file ("crossdomain.xml" in Flash and "clientaccesspolicy.xml" in Silverlight) defines a whitelist of domains from which a server is allowed to make cross-domain requests. When making a cross-domain request, the Flash or Silverlight client will first look for the policy file on the target server. If it is found, and the domain hosting the application is explicitly allowed to make requests, the request is made. Therefore, if a cross-domain policy file includes domains that should not be trusted, such as when using wildcards, then the application could be attacked by these untrusted domains. An overly permissive policy file allows many of the same attacks seen in Cross-Site Scripting (CWE-79). Once the user has executed a malicious Flash or Silverlight application, they are vulnerable to a variety of attacks. The attacker could transfer private information, such as cookies that may include session information, from the victim's machine to the attacker. The attacker could send malicious requests to a web site on behalf of the victim, which				CWE-942

		could be especially dangerous to the site if the victim has administrator privileges to manage that site. In many cases, the attack can be launched without the victim even being aware of it.				
T39	buffer overflow	Buffer overflow errors are characterized by the overwriting of memory spaces of the background web process, which should have never been modified intentionally or unintentionally.		SRC4	WASC-07	CWE-120
T40	FORMAT STRING ERROR	A format string error occurs when an input string is interpreted by the application as a command.		SRC4	WASC-06	CWE-134
T41	Password Autocomplete in browser	AUTOCOMPLETE attribute is not disabled in HTML FORM/INPUT element containing password type input. Passwords may be stored in browsers and retrieved.		SRC4		
T42	Content-Type Header Missing	Content-Type header missing.				

Appendix 7. Results of the security review

In the following we present results of the security review performed for the SPECS artifacts. The review is defined in deliverable D4.5.2 and is based on the Application Security Verification Standard (ASVS 2.0) proposed by OWASP[13]. The checklist comprises a set of requirements that should be addressed during the development stage in order to assure secure software. SPECS developers have assessed SPECS artifacts (SLA Platform (SLAP), Negotiation module (NEG), Enforcement module (ENF), Monitoring module (MON), Vertical Layer (VL), Enabling Platform, default SPECS Application, and the EMC Testbed) by verifying whether each requirement on the defined checklist has been covered or not (these requirements are labelled as ✓ or ×, respectively). The results are presented in the following tables, which group defined requirements across different security areas.

Note that the questionnaire has been developed to be used for the overall applications; therefore some requirements may not be applicable to all components/module; in this case we report the *N label*. Moreover, in some other cases the replies depend on the configuration and on the adoption of specific layers. For example, the adoption of the TLS for all communication (requirement SC69) is a matter of deployment and does not affect the development process. For such cases we use the *label D*.

[13] The Open Web Application Security Project, "*Application Security Verification Standard (2014)*", 2014. [Online]. Available online, https://www.owasp.org/images/5/58/OWASP_ASVS_Version_2.pdf, last accessed in March 2016.

SPECS secure web application requirements																									
			SLAP				NEG			ENF				MON					VL						
ID	Requirement	Enabling Platform	SLA Manager	Service Manager	Metric Catalogue	Interoperability Layer	SLO Manager	Supply Chain Manager	Security Reasoner	Planning	Implementation	Diagnosis	RDS	Event Hub	MoniPoli Filter	CTP	Event archiver	Nmap	Auditing	Security Tokens	Credential Service	User Manager	SPECS Application	EMC Testbed
a. Authentication verification requirements																									
SC1	Verify all resources require authentication except those specifically intended to be public (Principle of complete mediation).	✓	✓	✓	✓	✓	✓	N	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SC2	Verify all authentication controls are enforced on the server side.	✓	✓	✓	✓	✓	N	N	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SC3	Verify all authentication controls (including libraries that call external authentication services) have a centralized implementation.	N	✓	✓	✓	✓	N	N	✓	✓	✓	✓	✓	N	N	✓	N	N	✓	✓	✓	✓	✓	✓
SC4	Verify all authentication controls fail securely to ensure attackers cannot log in.	✓	✓	✓	✓	✓	N	N	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SC5	Verify all account identity authentication functions (such as registration, update profile, forgot username, forgot password, disabled / lost token, help desk or IVR) that might regain access to the account are at least as resistant to attack as the primary authentication mechanism.	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	×	✓
SC6	Verify users can safely change their credentials using a mechanism that is at least as resistant to attack as the primary authentication mechanism.	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	×	✓
SC7	Verify that all authentication decisions are logged. This should include requests with missing required information, needed for security investigations.	N	N	N	N	N	N	N	N	N	N	N	N	N	N	✓	N	N	N	✓	N	N	×	✓
SC8	Verify that account passwords are salted using a salt that is unique to that account (e.g., internal user ID, account creation) and use bcrypt, scrypt or PBKDF2 before storing the password.	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	✓	N
SC9	Verify that credentials, and all other identity information handled by the application(s), do not traverse unencrypted or weakly encrypted links.	D	D	D	D	D	N	N	D	N	N	N	N	D	D	✓	D	D	N	✓	D	D	D	✓

SC10	Verify that the forgotten password function and other recovery paths do not reveal the current password and that the new password is not sent in clear text to the user.	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	✓
SC11	Verify that username enumeration is not possible via login, password reset, or forgot account functionality.	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	✓	✓
SC12	Verify there are no default passwords in use for the application framework or any components used by the application (such as “admin/password”).	N	N	N	N	N	N	N	N	N	N	N	N	N	N	✓	N	N	N	N	N	N	✓	✓
SC13	Verify that a resource governor is in place to protect against vertical (a single account tested against all possible passwords) and horizontal brute forcing (all accounts tested with the same password e.g. “Password1”). A correct credential entry should incur no delay. Both these governor mechanisms should be active simultaneously to protect against diagonal and distributed attacks.	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	×	✓
SC14	Verify that all authentication credentials for accessing services external to the application are encrypted and stored in a protected location (not in source code).	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	✓	✓
SC15	Verify that forgot password and other recovery paths send a link including a time-limited activation token rather than the password itself. Additional authentication based on soft-tokens (e.g. SMS token, native mobile applications, etc.) can be required as well before the link is sent over.	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	✓
SC16	Verify that forgot password functionality does not lock or otherwise disable the account until after the user has successfully changed their password. This is to prevent valid users from being locked out.	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	✓
SC17	Verify that there are no shared knowledge questions/answers (so called "secret" questions and answers).	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	✓	✓
b. Access control verification requirements																									
SC18	Verify that users can only access secured functions or services for which they possess specific authorization.	✓	✓	✓	✓	✓	N	N	✓	✓	✓	✓	✓	✓	N	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SC19	Verify that users can only access secured URLs for which they possess specific authorization.	N	✓	✓	✓	✓	N	N	✓	✓	✓	✓	✓	✓	N	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SC20	Verify that users can only access secured data files for which they possess specific authorization.	N	✓	✓	✓	✓	N	N	✓	✓	✓	✓	✓	✓	N	N	✓	✓	✓	✓	✓	✓	✓	✓	✓
SC21	Verify that direct object references are protected, such that only	N	✓	✓	✓	✓	N	N	✓	✓	✓	✓	✓	✓	N	N	N	✓	✓	✓	✓	✓	✓	✓	N

	authorized objects or data are accessible to each user (for example, protect against direct object reference tampering).																							
SC22	Verify that access controls fail securely.	✓	✓	✓	✓	✓	N	N	✓	✓	✓	✓	✓	✓	N	✓	✓	✓	✓	✓	✓	✓	✓	✓
SC23	Verify that all user and data attributes and policy information used by access controls cannot be manipulated by end users unless specifically authorized.	✓	✓	✓	✓	✓	N	N	✓	✓	✓	✓	✓	✓	N	✓	✓	✓	✓	✓	✓	✓	✓	✓
SC24	Verify that all access controls are enforced on the server side.	✓	✓	✓	✓	✓	N	N	✓	✓	✓	✓	✓	✓	N	✓	✓	✓	✓	✓	✓	✓	✓	✓
SC25	Verify that there is a centralized mechanism (including libraries that call external authorization services) for protecting access to each type of protected resource.	N	✓	✓	✓	✓	N	N	✓	✓	✓	✓	✓	N	N	✓	N	N	✓	✓	✓	✓	✓	✓
SC26	Verify that all access control decisions are logged and all failed decisions are logged.	×	×	×	×	×	N	N	×	✓	✓	✓	✓	×	N	✓	×	✓	✓	✓	×	×	×	✓
SC27	Aggregate access control protection – verify the system can protect against aggregate or continuous access of secured functions, resources, or data. For example, possibly by the use of a resource governor to limit the number of edits per hour or to prevent the entire database from being scraped by an individual user.	×	×	×	×	×	N	N	×	×	×	×	×	×	N	N	×	×	×	×	×	×	×	N
c. Malicious input handling verification requirements																								
SC28	Verify that the runtime environment is not susceptible to buffer overflows, or that security controls prevent buffer overflows.	×	×	×	×	×	N	×	×	×	×	×	×	×	×	✓	×	×	×	×	×	×	×	✓
SC29	Verify that all input validation failures result in input rejection.	✓	×	×	×	×	✓	✓	×	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	×	×	×	✓
SC30	Verify that a character set, such as UTF-8, is specified for all sources of input.	×	×	×	×	×	✓	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×
SC31	Verify that all input validation or encoding routines are performed and enforced on the server side.	×	×	×	×	×	✓	✓	×	×	×	×	×	✓	✓	✓	✓	✓	×	×	×	×	×	✓
SC32	Verify that a single input validation control is used by the application for each type of data that is accepted.	×	×	×	×	×	✓	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	✓
SC33	Verify that all input validation failures are logged.	×	×	×	×	×	N	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	✓
SC34	Verify that all input data is canonicalized for all downstream decoders or interpreters prior to validation.	×	×	×	×	×	N	×	×	×	×	×	×	×	×	✓	×	×	×	×	×	×	×	✓
SC35	Verify that the runtime environment is not susceptible to SQL Injection, or that security controls prevent SQL Injection.	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	N	N	N	N	N	✓	✓	✓	✓	✓	N
SC36	Verify that the runtime environment is not susceptible to OS Command	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	N	N	N	N	N	✓	✓	✓	✓	✓	✓

	Injection, or that security controls prevent OS Command Injection.																							
SC37	Verify that the runtime environment is not susceptible to XML External Entity attacks or that security controls prevents XML External Entity attacks.	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	N	✓	N	N	N	✓	✓	✓	✓	✓	N
SC38	Verify that the runtime environment is not susceptible to XML Injections or that security controls prevents XML Injections.	N	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	N	✓	N	N	N	✓	✓	✓	✓	✓	N
SC39	If the application framework allows automatic mass parameter assignment (also called automatic variable binding) from the inbound request to a model, verify that security sensitive fields such as "accountBalance", "role" or "password" are protected from malicious automatic binding.	N	✓	✓	✓	✓	N	✓	✓	✓	✓	✓	✓	N	N	N	N	✓	✓	✓	✓	✓	✓	N
SC40	Verify that for each type of output encoding/escaping performed by the application, there is a single security control for that type of output for the intended destination.	N	×	×	×	×	N	×	×	×	×	×	×	N	N	✓	N	×	×	×	×	×	×	N
d. Cryptography at rest verification requirements																								
SC41	Verify that all cryptographic functions used to protect secrets from the application user are implemented server side.	N	N	N	N	N	N	N	N	N	N	N	N	N	N	✓	N	N	N	N	N	N	N	✓
SC42	Verify that all cryptographic modules fail securely.	N	N	N	N	N	N	N	N	N	N	N	N	N	N	✓	N	N	N	N	N	N	N	✓
SC43	Verify that access to any master secret(s) is protected from unauthorized access (A master secret is an application credential stored as plaintext on disk that is used to protect access to security configuration information).	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
SC44	Verify that all random numbers, random file names, random GUIDs, and random strings are generated using the cryptographic module's approved random number generator when these random values are intended to be unguessable by an attacker.	N	N	N	N	N	N	N	N	N	N	N	N	N	N	✓	N	N	N	N	N	N	N	✓
SC45	Verify that cryptographic modules used by the application have been validated against FIPS 140-2 or an equivalent standard.	N	N	N	N	N	N	N	N	N	N	N	N	N	N	×	N	N	N	N	N	N	N	✓
SC46	Verify that cryptographic modules operate in their approved mode according to their published security policies.	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	✓
SC47	Verify that there is an explicit policy for how cryptographic keys are managed (e.g., generated, distributed, revoked, expired). Verify that this policy is properly enforced.	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	✓
e. Error handling and logging verification requirements																								

SC48	Verify that the application does not output error messages or stack traces containing sensitive data that could assist an attacker, including session id and personal information.	✗	✗	✗	✗	✗	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗	✗	✓
SC49	Verify that all error handling is performed on trusted devices	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SC50	Verify that all logging controls are implemented on the server.	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SC51	Verify that error handling logic in security controls denies access by default.	N	N	N	N	N	N	N	N	N	N	N	N	N	N	✓	N	N	N	N	N	N	N	✓
SC52	Verify security logging controls provide the ability to log both success and failure events that are identified as security-relevant.	✗	✗	✗	✗	✗	N	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✓
SC53	Verify that each log event includes: a timestamp from a reliable source, the severity level of the event, an indication that this is a security relevant event (if mixed with other logs), the identity of the user that caused the event (if there is a user associated with the event), the source IP address of the request associated with the event, whether the event succeeded or failed, and a description of the event.	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓
SC54	Verify that all events that include untrusted data will not execute as code in the intended log viewing software.	N	N	N	N	N	N	N	N	N	N	N	N	N	N	✓	N	N	N	N	N	N	N	✓
SC55	Verify that security logs are protected from unauthorized access and modification.	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	✓
SC56	Verify that there is a single application-level logging implementation that is used by the software.	N	N	N	N	N	✓	N	N	N	N	N	N	✓	✓	✓	✓	✓	N	N	N	N	N	✓
SC57	Verify that the application does not log application-specific sensitive data that could assist an attacker, including user's session identifiers and personal or sensitive information. The length and existence of sensitive data can be logged.	N	N	N	N	N	N	N	N	N	N	N	N	N	N	✓	N	✓	N	N	N	N	N	✓
SC58	Verify that a log analysis tool is available which allows the analyst to search for log events based on combinations of search criteria across all fields in the log record format supported by this system.	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
SC59	Verify that all non-printable symbols and field separators are properly encoded in log entries, to prevent log injection.	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
SC60	Verify that log fields from trusted and untrusted sources are distinguishable in log entries.	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
SC61	Verify that logging is performed before executing the transaction. If	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N

	logging was unsuccessful (e.g. disk full, insufficient permissions) the application fails safe. This is for when integrity and non-repudiation are a must.																							
f. Data protection verification requirements																								
SC62	Verify that the list of sensitive data processed by this application is identified, and that there is an explicit policy for how access to this data must be controlled, and when this data must be encrypted (both at rest and in transit). Verify that this policy is properly enforced.	N	N	N	N	N	N	N	N	N	N	N	N	N	N	✓	N	N	N	N	N	N	N	N
SC63	Verify that all sensitive data is sent to the server in the HTTP message body (i.e., URL parameters are never used to send sensitive data).	✓	✓	✓	✓	✓	N	N	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	N
SC64	Verify that all cached or temporary copies of sensitive data stored on the server are protected from unauthorized access or purged/invalidated after the authorized user accesses the sensitive data.	✓	✓	✓	✓	✓	N	✓	✓	✓	✓	✓	✓	✓	✓	N	✓	✓	✓	✓	✓	✓	✓	N
SC65	Verify that there is a method to remove each type of sensitive data from the application at the end of its required retention period.	N	×	×	×	×	N	×	×	×	×	×	×	N	N	✓	✓	×	×	×	×	×	×	N
SC66	Verify the application has the ability to detect and alert on abnormal numbers of requests for information or processing high value transactions for that user role, such as screen scraping, automated use of web service extraction, or data loss prevention. For example, the average user should not be able to access more than 5 records per hour or 30 records per day, or add 10 friends to a social network per minute.	×	×	×	×	×	N	N	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	✓
g. Communications security verification requirements																								
SC67	Verify that a path can be built from a trusted CA to each Transport Layer Security (TLS) server certificate, and that each server certificate is valid.	D	D	D	D	D	N	N	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	N
SC68	Verify that failed TLS connections do not fall back to an insecure HTTP connection.	D	D	D	D	D	N	N	D	D	D	D	D	D	D	✓	D	D	D	D	D	D	D	N
SC69	Verify that TLS is used for all connections (including both external and backend connections) that are authenticated or that involve sensitive data or functions.	D	D	D	D	D	N	N	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	N
SC70	Verify that backend TLS connection failures are logged.	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
SC71	Verify that certificate paths are built and verified for all client certificates using configured trust anchors and revocation information.	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
SC72	Verify that all connections to external systems that involve sensitive	N	✓	✓	✓	✓	N	N	✓	✓	✓	✓	✓	✓	✓	N	✓	✓	✓	✓	✓	✓	✓	N

	information or functions are authenticated.																							
SC73	Verify that all connections to external systems that involve sensitive information or functions use an account that has been set up to have the minimum privileges necessary for the application to function properly.	N	×	×	×	×	N	N	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	N
SC74	Verify that there is a single standard TLS implementation that is used by the application that is configured to operate in an approved mode of operation. (See http://csrc.nist.gov/groups/STM/cmvp/documents/fips140-2/FIPS1402IG.pdf).	N	D	D	D	D	N	N	D	D	D	D	D	N	N	×	N	N	D	D	D	D	D	N
SC75	Verify that specific character encodings are defined for all connections (e.g., UTF-8).	×	×	×	×	×	N	N	×	×	×	×	×	×	×	✓	×	×	×	×	×	×	×	N
h. HTTP security verification requirements																								
SC76	Verify that the application accepts only a defined set of HTTP request methods, such as GET and POST and unused methods are explicitly blocked.	✓	✓	✓	✓	✓	✓	N	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SC77	Verify that every HTTP response contains a content type header specifying a safe character set (e.g., UTF-8).	✓	✓	✓	✓	✓	✓	N	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SC78	Verify that HTTP headers in both requests and responses contain only printable ASCII characters.	✓	×	×	×	×	✓	N	×	×	×	×	×	✓	✓	✓	✓	✓	×	×	×	×	×	✓
SC79	Verify that HTTP headers added by a frontend (such as X-Real-IP), and used by the application, cannot be spoofed by the end user.	×	×	×	×	×	✓	N	×	×	×	×	×	×	×	N	×	×	×	×	×	×	×	✓
SC80	Verify that the HTTP headers do not expose detailed version information of system components.	✓	×	×	×	×	✓	N	×	×	×	×	×	✓	✓	✓	✓	✓	×	×	×	×	×	✓
i. Malicious controls verification requirements																								
SC81	Verify that no malicious code is in any code that was either developed or modified in order to create the application.	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SC82	Verify that the integrity of interpreted code, libraries, executables, and configuration files is verified using checksums or hashes.	×	×	×	×	×	✓	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	✓
SC83	Verify that all code implementing or using authentication controls is not affected by any malicious code.	✓	✓	✓	✓	✓	N	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SC84	Verify that all code implementing or using access controls is not affected by any malicious code.	✓	✓	✓	✓	✓	N	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SC85	Verify that all input validation controls are not affected by any malicious	✓	D	D	D	D	✓	D	D	D	D	D	D	✓	✓	✓	✓	✓	D	D	D	D	D	✓

	code.																							
SC86	Verify that all code implementing or using output validation controls is not affected by any malicious code.	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SC87	Verify that all code supporting or using a cryptographic module is not affected by any malicious code.	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
SC88	Verify that all code implementing or using error handling and logging controls is not affected by any malicious code.	N	N	N	N	N	N	N	N	N	N	N	N	N	N	✓	N	N	N	N	N	N	N	✓
SC89	Verify all malicious activity is adequately sandboxed.	×	×	×	×	×	N	×	×	×	×	×	×	×	×	✓	×	×	×	×	×	×	×	✓
SC90	Verify that sensitive data is rapidly sanitized from memory as soon as it is no longer needed and handled in accordance to functions and techniques supported by the framework/library/operating system.	×	×	×	×	×	N	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	✓
j. Business logic verification requirements																								
SC91	Verify the application processes or verifies all high value business logic flows in a trusted environment, such as on a protected and monitored server.	✓	✓	✓	✓	✓	N	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SC92	Verify the application does not allow spoofed high value transactions, such as allowing Attacker User A to process a transaction as Victim User B by tampering with or replaying session, transaction state, transaction or user IDs.	N	×	×	×	×	N	×	×	×	×	×	×	N	N	N	N	N	×	×	×	×	×	✓
SC93	Verify the application does not allow high value business logic parameters to be tampered with, such as (but not limited to): price, interest, discounts, PII, balances, stock IDs, etc.	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
SC94	Verify the application has defensive measures to protect against repudiation attacks, such as verifiable and protected transaction logs, audit trails or system logs, and in highest value systems real time monitoring of user activities and transactions for anomalies.	×	×	×	×	×	N	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	✓
SC95	Verify the application protects against information disclosure attacks, such as direct object reference, tampering, session brute force or other attacks.	✓	✓	✓	✓	✓	N	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SC96	Verify the application has sufficient detection and governor controls to protect against brute force (such as continuously using a particular function) or denial of service attacks.	×	×	×	×	×	N	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	✓
SC97	Verify the application has sufficient access controls to prevent elevation of privilege attacks, such as allowing anonymous users from accessing	N	×	×	×	×	N	×	×	×	×	×	×	×	N	✓	×	✓	×	×	×	×	×	✓

	secured data or secured functions, or allowing users to access each other's details or using privileged functions.																							
SC98	Verify the application will only process business logic flows in sequential step order, with all steps being processed in realistic human time, and not process out of order, skipped steps, process steps from another user, or too quickly submitted transactions.	N	×	×	×	×	✓	×	×	×	×	×	×	N	N	✓	N	N	×	×	×	×	×	✓
SC99	Verify the application has additional authorization (such as step up or adaptive authentication) for lower value systems, and / or segregation of duties for high value applications to enforce anti-fraud controls as per the risk of application and past fraud.	N	×	×	×	×	N	×	×	×	×	×	×	N	N	N	N	N	×	×	×	×	×	✓
SC100	Verify the application has business limits and enforces them in a trusted location (as on a protected server) on a per user, per day or daily basis, with configurable alerting and automated reactions to automated or unusual attack. Examples include (but not limited to): ensuring new SIM users don't exceed $10 per day for a new phone account, a forum allowing more than 100 new users per day or preventing posts or private messages until the account has been verified, a health system should not allow a single doctor to access more patient records than they can reasonably treat in a day, or a small business finance system allowing more than 20 invoice payments or $1000 per day across all users. In all cases, the business limits and totals should be reasonable for the business concerned. The only unreasonable outcome is if there are no business limits, alerting or enforcement.	N	✓	✓	✓	✓	N	✓	✓	✓	✓	✓	✓	N	N	N	N	N	✓	✓	✓	✓	✓	✓
k. Files and resources verification requirements																								
SC101	Verify that file names and path data obtained from untrusted sources is canonicalized to eliminate path traversal attacks.	N	×	×	×	×	N	×	×	×	×	×	×	N	N	N	N	N	×	×	×	×	×	N
SC102	Verify that files obtained from untrusted sources are scanned by antivirus scanners to prevent upload of known malicious content.	N	×	×	×	×	N	×	×	×	×	×	×	N	×	N	N	N	×	×	×	×	×	N
SC103	Verify that parameters obtained from untrusted sources are not used in manipulating filenames, pathnames or any file system object without first being canonicalized and input validated to prevent local file inclusion attacks.	✓	×	×	×	×	N	×	×	×	×	×	×	✓	✓	N	✓	✓	×	×	×	×	×	✓
SC104	Verify that parameters obtained from untrusted sources are canonicalized, input validated, and output encoded to prevent remote file	N	×	×	×	×	N	×	×	×	×	×	×	N	N	N	N	N	×	×	×	×	×	N

	inclusion attacks, particularly where input could be executed, such as header, source, or template inclusion																							
SC105	Verify that web or application server is configured by default to deny access to remote resources or systems outside the web or application server.	✓	✓	✓	✓	✓	✓	N	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SC106	Verify the application code does not execute uploaded data obtained from untrusted sources.	✓	✓	✓	✓	✓	N	N	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓