SOCCRATES

SOC & CSIRT Response to Attacks & Threats based on attack defense graphs Evaluation Systems

D6.3

SOCCRATES testing and verification report

Deliverable type:	Report
Contributing work packages:	WP6
Due date of deliverable:	31/01/2022
Submission date:	31/03/2022
Dissemination level:	PU

Responsible organisation:	TNO
Editor:	Frank Fransen
Revision:	1.0

Abstract	This deliverable describes the approach and results of the testing and verification of the SOCCRATES Platform. It includes a description of the testing environments, the tests and results.
Keywords:	SOCCRATES, testing, verification, integration testing, deployment testing

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no. 833481
Call H2020-SU-ICT-2018 • Innovation Action • Start date: September 1st, 2019

Management summary

This deliverable describes the approach and results of the testing and verification of the SOCCRATES Platform. It includes a description of the testing strategy, test environments, the testing plan and the results.

Within SOCCRATES many individual software tools are developed. To ensure the correctness of software with respect to the SOCCRATES Platform specifications and good quality of the software, testing and verification activities have been performed. In SOCCRATES the unit testing of a component and component internal integration testing is done within the work package responsible for the development of that component. Within SOCCRATES WP6 the focus was on the following two testing activities:

1. *Platform Integration testing* – Testing to verify that components can be integrated (*integration testing*) and testing to verify that the SOCCRATES platform can execute the SOCCRATES Use Cases as specified in D2.3.
2. *Platform deployment testing* – Preparation of all software for deployment and testing to verify that SOCCRATES Platform can be deployed as a whole.

Integration testing has been segmented into tests for triggering each of the SOCCRATES use cases, tests for Integration Activiti with Cortex in the Orchestration and Integration Engine, tests for integration Cortex Analysers / Responders with specific components, tests for integration with the Shadowserver API, and tests for the Web Front-End. All the SOCCRATES integration tests have been successfully completed.

For the deployment testing a staging environment has been setup with a so called Continues Integration and Continuous Deployment (CI/CD) pipeline. It automatically performs tests on the software provided by the partners and deploys each of the SOCCRATES platform components within the staging environment. The execution and provisioning of the services provided by each of the SOCCRATES platform components are automatically monitored by a tool called Portainer. All SOCCRATES components have been successfully deployed in the staging environment. The SOCCRATES partners have been provided access to the web-based services provided by the SOCCRATES components, and have successfully verified correct deployment of their SOCCRATES component.

Contributor(s)	Jesús Villalobos Nieto (ATOS) Susana González Zarzosa (ATOS) Martin Eian (mnemonic) Frank Fransen (TNO) Sebastiaan Tesink (TNO) Federico Falconeri (TNO) Erik Ringdahl (FRS) Christophe Kiennert (IMT) Paul-Henri Mignot (IMT)

Reviewer(s)	Paul Smith (AIT) Reinder Wolthuis (TNO)

Security Assessment	**See deliverables table for security assessment requirements**
Approval Date	11/03/2022
Remarks	Minor comments, all adressed

TABLE OF CONTENTS

1 INTRODUCTION – RATIONALE OF THIS DOCUMENT 6

1.1 The SOCCRATES project 6
1.2 This deliverable 7

2 SOCCRATES TESTING STRATEGY 8

2.1 Testing levels & approach 8
2.2 Integration testing template 10
2.3 Test Environments 10
2.3.1 Integration testing environment 10
2.3.2 Deployment testing environment 11
2.4 Test planning 12

3 PLATFORM INTEGRATION TESTING 13

3.1 Trigging workflows 13
3.1.1 UC1: Response on Detected Ongoing attack 13
3.1.2 UC2: Response on Reception of New Cyber Threat Intelligence 15
3.1.3 UC3: Response on Discovery of New Vulnerable Assets 17
3.1.4 UC4: Response on Discovery of System Configuration Change 19
3.1.5 UC5: Response on Deployment of New Systems in Infrastructure 20
3.2 Integration Activiti – Cortex Analysers / Responders 22
3.3 Integration Cortex Analysers / Responders – Components 24
3.3.1 IMC Cortex Analyzer 25
3.3.2 ADG Cortex Analyser 25
3.3.3 BIA Cortex Analyser 26
3.3.4 RP Cortex Analyser 26
3.3.5 COA Cortex Analyser 26
3.3.6 AAD Cortex Responder 26
3.3.7 BIA Cortex Responder 27
3.3.8 AR Cortex Responder 27
3.4 Integration with SHS 27
3.5 Web Front End 27
3.5.1 Test phase 1 – Get frontend sources 28
3.5.2 Test phase 2 – Get data from Activiti through backend 29
3.5.3 Test phase 3 – Save data to DB through backend 30

4 PLATFORM DEPLOYMENT TESTING 33

4.1 Making the deployment declarative 33
4.2 Continuous Integration and Continuous Deployment (CI/CD) 33
4.3 Container monitoring 36
4.4 Log4j testing 37

5 OVERALL RESULTS AND CONCLUSION 39

5.1 Test Execution 39
5.2 Overall test results 40

6 REFERENCES 41
7 ABBREVIATIONS 42
APPENDIX A. INTEGRATION TEST CASES 44

1 Introduction – Rationale of this document

This section introduces the SOCCRATES project and defines the goals of this deliverable.

1.1 The SOCCRATES project

SOCCRATES (SOC & CSIRT Response to Attacks & Threats based on attack defence graphs Evaluation Systems) is an EU funded project under the Horizon2020 programme that has the following main challenge:

How can SOC and CSIRT operations effectively improve their capability in detecting and managing response to complex cyber-attacks and emerging threats, in complex and continuously evolving ICT infrastructures while there is a shortage of qualified cybersecurity talent?

The main objective of SOCCRATES is to develop and implement a security automation and decision support platform (‘**the SOCCRATES platform’)** that will significantly improve an organisation’s capability (usually implemented by a SOC and/or CSIRT) to quickly and effectively detect and respond to new cyber threats and ongoing attacks.

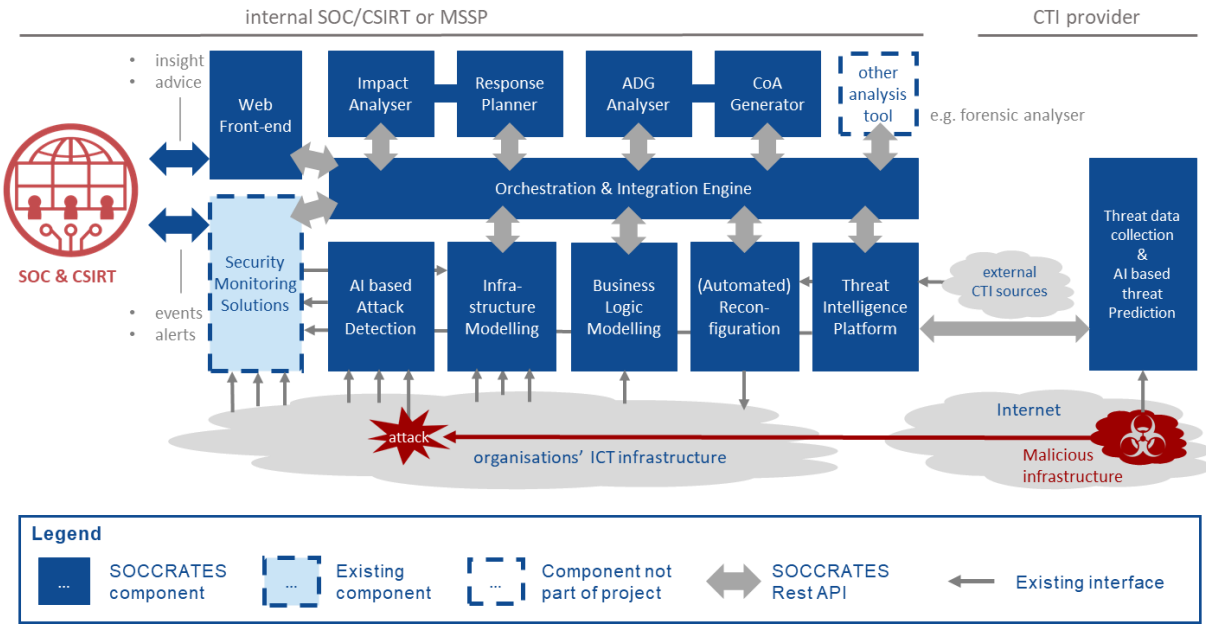


Figure 1-1 – The SOCCRATES platform

The SOCCRATES platform (see **Figure 1-1**) consists of an orchestrating function and a set of innovative components for automated infrastructure modelling, attack detection, cyber threat intelligence utilization, threat trend prediction, and automated analysis using attack defence graphs and business impact modelling to aid human analysis and decision making on response actions, and enable the execution of defensive actions at machine-speed.

SOCCRATES has the following concrete project objectives:

1. Deliver the SOCCRATES platform consisting of an orchestration function and a unique integration of innovative background solutions that seamlessly work together.
2. Show that the SOCCRATES platform can improve SOC operations by evaluating the SOCCRATES platform in two diverse real-life pilot environments.
3. Examine and illustrate the benefits of automation for selected SOC activities to help manage the cyber security skills gap in organizations.
4. Prepare for successful exploitation by the SOCCRATES partners of the individual innovated components and the integrated SOCCRATES platform in commercial products that are offered to the market and are available for the European (business) community.

Please visit www.soccrates.eu for more information on the SOCCRATES project.

1.2 This deliverable

This deliverable is the third report of WP6, framed mostly on results of Task 6.3 testing and verification[1]. The goal of this deliverable is to report on the approach and results of the testing and verification of the SOCCRATES Platform that has been performed before delivering the SOCCRATES platform components to the pilot environments (for phase 2 pilots). Testing includes verification of the functionality of individual components and of the SOCCRATES platform as a whole.
The testing and verification reported on in this deliverable is on platform integration testing and on platform deployment testing.

1.3 Structure of this deliverable

First the SOCCRATES testing strategy is explained in chapter 2. Chapter 3 reports on the platform integration testing activities and results, and in chapter 4 on platform deployment testing activities and results. Finally, in chapter 5 the overall results and conclusion of the SOCCRATES testing strategy is described.

[1] Task 6.3 was originally called Testing and Validation. As validation takes place in the pilots executed in WP7, the focus of the task 6.3 was on verification (i.e. the process of evaluating a system in a development phase to find out whether they meet the specified requirements).

2 SOCCRATES testing strategy

Within SOCCRATES many individual software tools are developed. To ensure the correctness of software with respect to the SOCCRATES Platform specifications, i.e. D2.3 [1], and good quality of the software, testing and verification activities have been performed. There are however many different software quality aspects, levels of testing (unit, integration, system, acceptance), and approaches.

In the next section, the SOCCRATES testing approach is set forth, and the focus of the testing performed in work package 6 is described. Next the integration testing template is presented and test environments are described. Finally, the testing plan is briefly described.

2.1 Testing levels & approach

As introduced above there are different levels where software testing can take place. Typically, the following four levels are distinguished:

1. *Unit testing* – refers to testing a specific section of the code, usually at a functional or component level.
2. *Integration testing* – refers to testing of the interfaces between functions or components against the design specification.
3. *System testing* – refers to the testing of the system as a whole, and to verify that the system meets its requirements.
4. *Acceptance testing* – refers to tests to determine if the requirements of a specification or contract are met. It includes, among others, user acceptance testing and operational acceptance testing.

Within SOCCRATES WP6, and this deliverable, only the first three levels of testing are addressed. Acceptance testing is partly addressed in the validation activities at the pilot sites, performed by WP7.

In SOCCRATES we can distinguish between two levels of unit testing. The unit testing at the SOCCRATES component level as depicted in the SOCCRATES platform component architecture, see **Figure 1-1**, and unit testing at the tools level that are combined to create a SOCCRATES component. The AI based Attack Detection (AAD) for instance consists of several individual anomaly detection software tools (e.g. DNS Ninja, ABC tool, Aminer, LADS), REASENS, Elastic, Kafka, Logstash, etc. On the AAD component one can do unit testing, integration testing and system testing. As WP4 is responsible for the development of SOCCRATES component AAD, we decided that WP4 is responsible for performing these tests on the AAD component. This approach is followed for all the SOCCRATES components. In other words, from SOCCRATES platform perspective unit testing a SOCCRATES component is performed by the individual work package that developed the component.

WP6 focusses on the following two testing activities:

- *Platform Integration testing* – Testing to verify that components can be integrated (*2. integration testing*) and testing to verify that the SOCCRATES platform can execute the SOCCRATES Use Cases as specified in D2.3 [1] *(3. system testing).*
- *Platform deployment testing* – Preparation of all software for deployment, and testing to verify that SOCCRATES Platform can be deployed as a whole. Note that *deployment testing* is not one of the four levels of software testing introduced at the beginning of the section. As the

SOCCRATES components are developed by different partners, this testing activity focusses on testing the combined deployment of all components in a single staging environment.

For platform Integration testing, the Orchestration and Integration Engine (OIE) plays a central role. The OIE consists of open source tools Activiti for Orchestration and Cortex for Integration. Activiti is an open source workflow engine that runs the workflows for the SOCCRATES Use Cases. Activiti provides a REST-API to trigger the workflow for SOCCRATES use cases. Activiti interacts with other SOCCRATES components via Cortex. Cortex is an open source project created by TheHive Project[2]. Cortex acts as a broker to communicate with individual security tools. Cortex provides a REST-API that can be used to interact with Cortex and invoke any of the security tools that are interconnected with it. Cortex allows to interact with external tools through two types of connectors:

- *Analysers* allow to invoke external tools by requesting some analysis or action and providing some data in the request. In general, these analysers provide a response with a format that depends on the type of tool.
- *Responders* allow to trigger some action in an external tool, without the need of receiving any observable in the response.

Integration testing can be divided into the following testing activities (see Figure 2-1):

A. Integration testing of triggering
B. Integration testing between Activiti and Cortex
C. Integration testing between Cortex Analysers/Responders and SOCCRATES Component

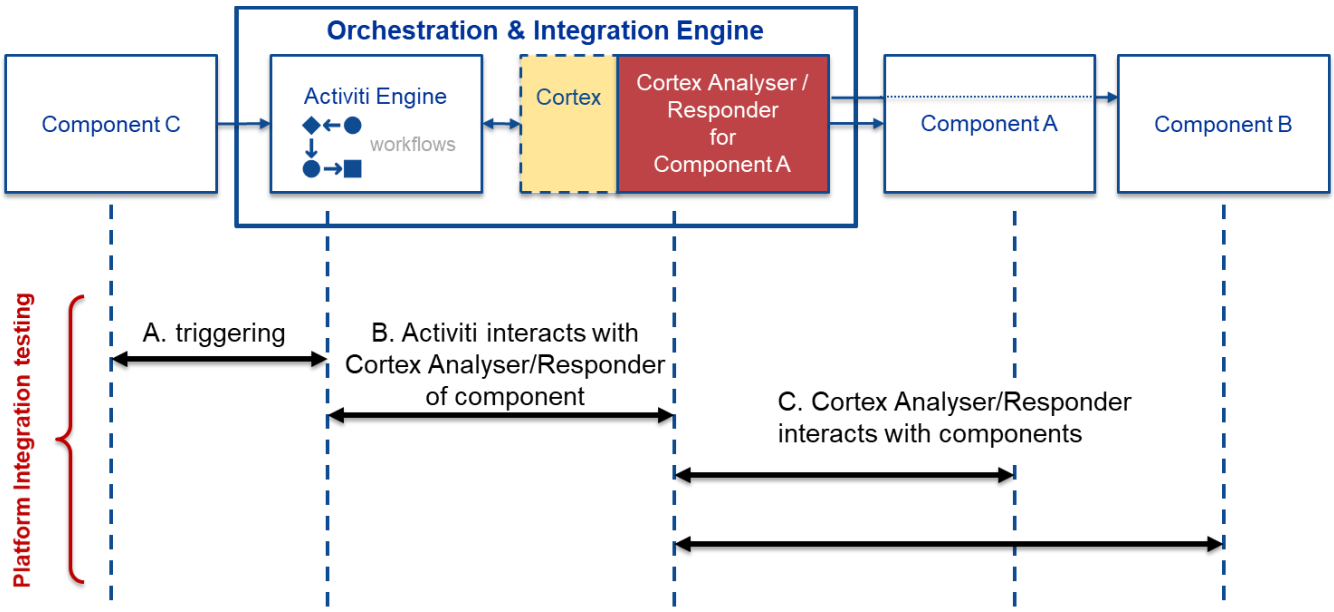


Figure 2-1 Platform integration testing parts.

Platform Integration testing is further described in chapter 3.

[2] https://github.com/TheHive-Project/Cortex

SOCCRATES uses CI/CD for platform deployment. Platform deployment testing thus relies heavily on automation within the CI/CD pipeline. Platform deployment testing is further described in chapter 4.

2.2 Integration testing template

Since platform integration testing depends on multiple parties collaborating, it is wise to define a testing approach. SOCCRATES defined the following template to specify the specific integration testing that will be performed.

Table 2.1. SOCCRATES integration testing template

Test Case ID		**Components**	
Type of testing	Integration		
Description			
Specification		**Priority**	Low/Medium/High
Prepared by		**Tested by**	
Pre-condition(s)			
Test steps			
1			
2			
3			
Input data			
Expected result			
Test Case Result	Success / Failed*)		
	If failed, why …		

2.3 Test Environments

2.3.1 Integration testing environment

The integration testing environment is distributed through servers located at the different partner premises and connected to the same VPN. Figure 2-2 shows the different components included in the testing environment.

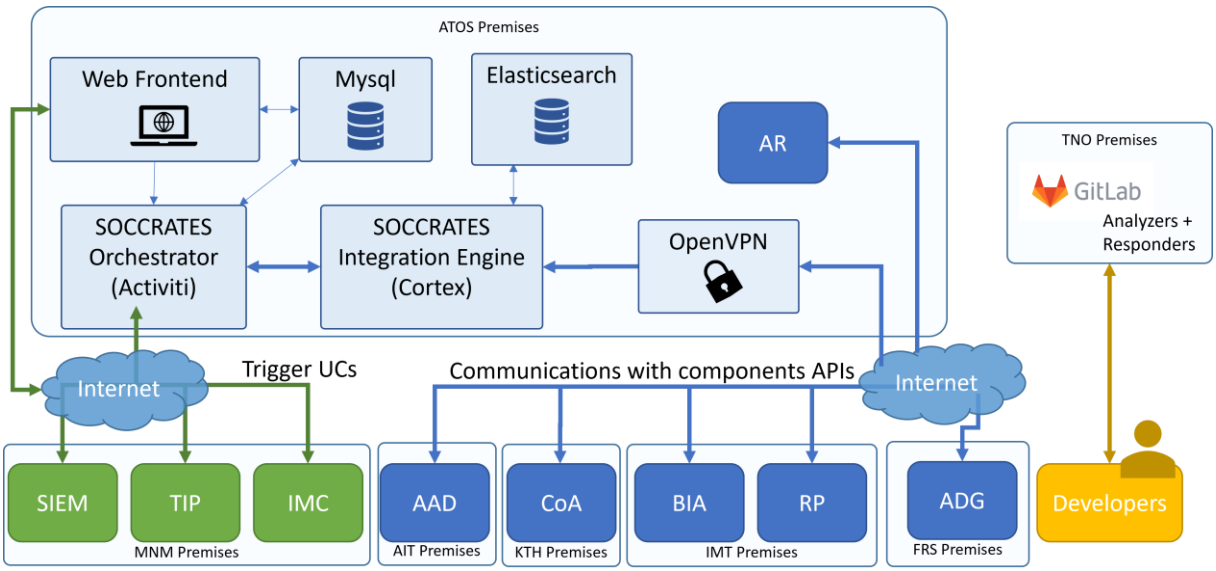


Figure 2-2 : SOCCRATES Integration Testing Environment

The description of the components included in the integration testing environment is almost the same as included in section 2.4 of D6.2 [3] with the following differences:

- There is a new component that is the SOCCRATES Orchestrator Web Frontend, deployed in Atos premises and connected with the same database used by the SOCCRATES Orchestrator. From the Web Front-end it is also possible to trigger the different UCs for testing purposes and access to the interfaces provided by other components. The access and authentication to this GUI is provided using the proxy Traefik[3] and Authelia[4] (more details are included in next section). More details about the web frontend are included in section 3.5.
- Initially an ftp server was used for the distribution to ATOS of the Cortex Analyzer and Responder (developed by other partners) for deployment in the SOCCRATES Integration Engine. This Vsftpd server has been substituted by the SOCCRATES GitLab repository deployed at TNO premises. This repository has been included as a git submodule in the SOCCRATES Integration Engine container.
- The deployment has also changed since currently all the components are deployed as Docker containers, as described in detail in the next section.

2.3.2 Deployment testing environment

To perform deployment testing, software must be deployed somewhere. For SOCCRATES deployment testing we relied on virtual machines from the TNO Research Cloud, an internal cloud environment based on OpenStack. This environment is connected to the TNO gitlab instance, where SOCCRATES software is stored. GitLab supports CI/CD pipelines that can be used to automatically build, test, and deploy software. Within the deployment test environment it is used for automatic deployment and testing of the software. Section 4.2 describes the CI/CD pipeline used to perform the deployment testing. In Figure 2-3 the SOCCRATES Deployment Testing Environment is depicted.

[3] https://doc.traefik.io/traefik/

[4] https://www.authelia.com/

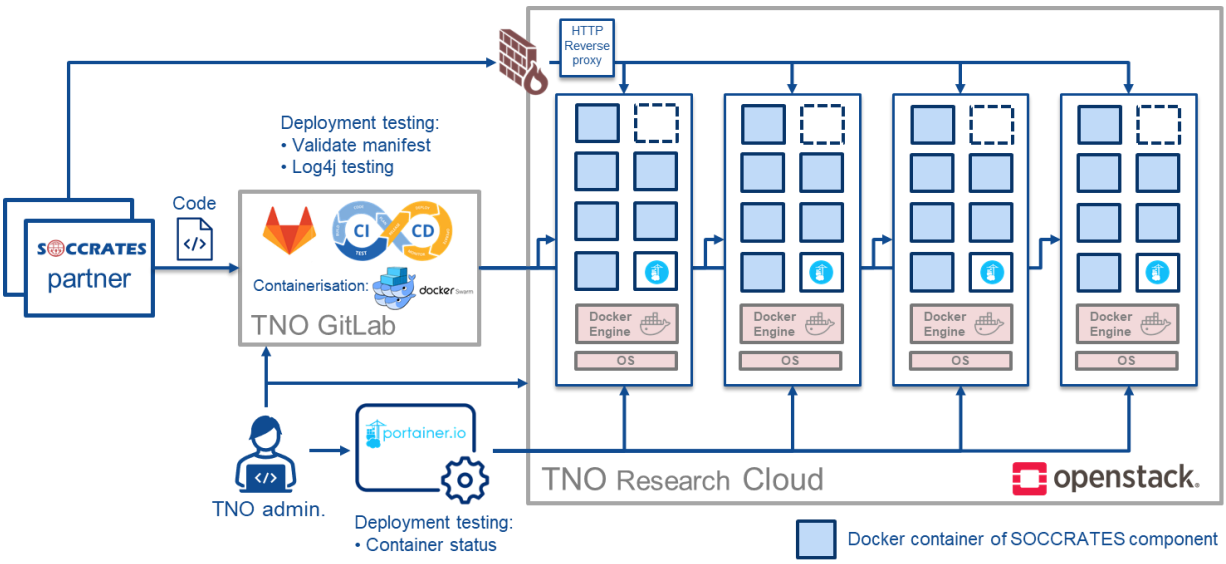


Figure 2-3 : SOCCRATES Deployment Testing Environment

Deploying the SOCCRATES components is not sufficient, because some of them need data from an infrastructure to work. Rather than actually setting up such an infrastructure, we settled on a simpler solution: *simulating it by producing mockup netflow and DNS data*. We developed an open source component, called NEtwork Data producer (Ned), to do this[5]. Ned can efficiently produce millions of records and push them to elasticsearch, from which they are then available for consumption by components like the AAD and the IMC.

2.4 Test planning

The goal of the SOCCRATES Platform testing was to be ready before the start of the second pilot phase. The plan for integration and deployment testing was therefore set to be finalised by the end of 2022, as depicted in Figure 2.4.

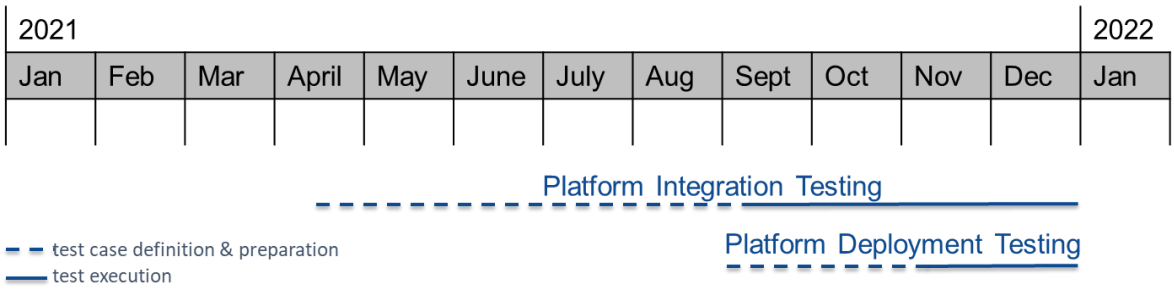


Figure 2.4: Task 6.3 planning.

[5] https://gitlab.com/cossas/ned

3 Platform Integration Testing

3.1 Trigging workflows

In this section, we report on integration part A in Figure 2-1. This refers to testing how the workflow of the different use cases (UC1 to UC5) is triggered in Activiti. Below, the triggering events used for testing each use case are described. As a result, a new orchestrator process will appear in the SOCCRATES Orchestrator web frontend associated to a specific type (incident, threat, vulnerability, configuration or system) for the workflow that is started. The details of the test cases are included in Appendix A.

3.1.1 UC1: Response on Detected Ongoing attack

The use case 1 should be triggered automatically from the SIEM when a new cyber security incident is detected providing the source and destination IP address of the ongoing attack as well as the source/destination ports, the severity of the attack, the date and the event type or tactic (i.e. if it is an initial access, a lateral movement or an exfiltration). In the case of Mnemonic site it will be done via Argus and in the case of Vattenfall via SIEM. Those integrations will be tested once the deployment of the SOCCRATES Platform in the pilots has been completed. For testing purposes, in this deliverable we only report the triggering of this use case from the web frontend, as defined in test case A.1 and shown in Figure 3-1.

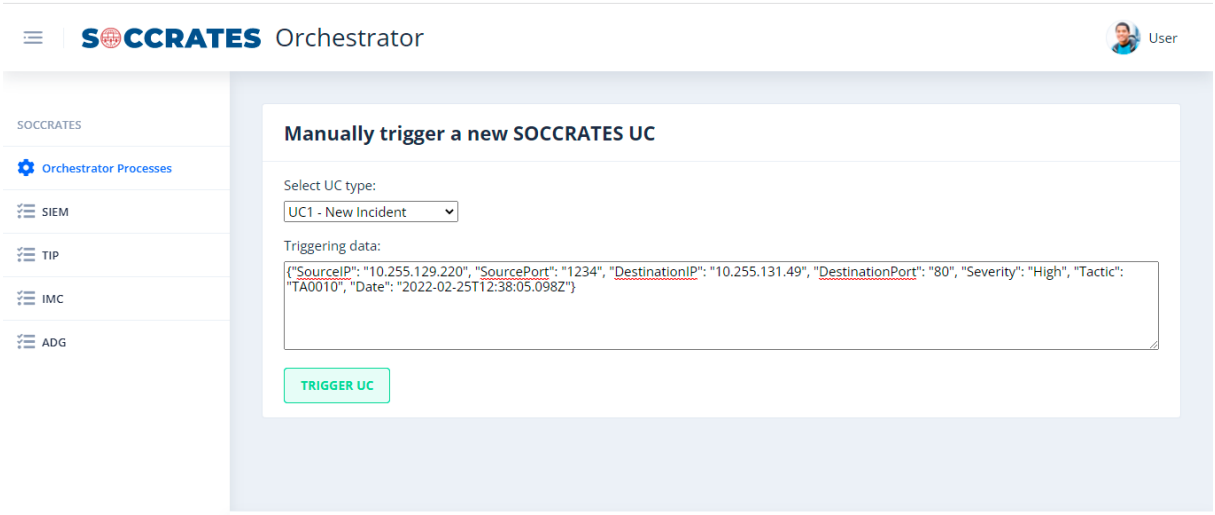


Figure 3-1 : Manual triggering of UC-1 from SOCCRATES Orchestrator Web Frontend

Once the use case is triggered, a notification appears in the web frontend (see Figure 3-2) and the new process started in the SOCCRATES Orchestrator appears with the assigned ID in the dashboard and the type “Incident (UC1)” (see Figure 3-3). The details of the triggering messages can be also visualized in the web frontend (see Figure 3-4).

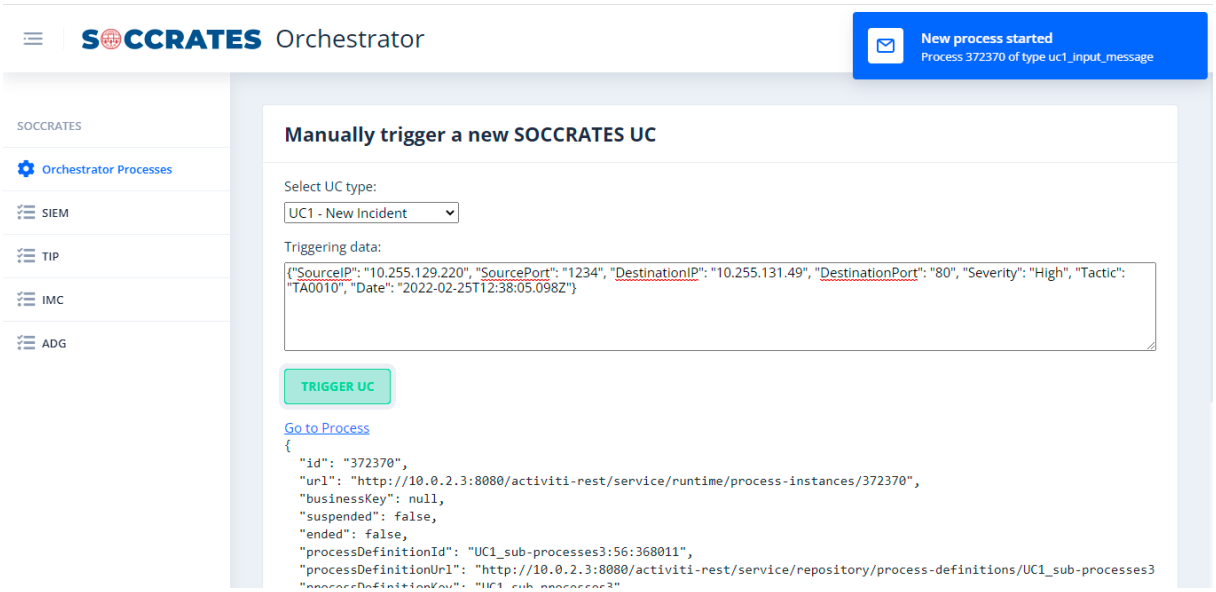


Figure 3-2 : UC1 notification triggered in SOCCRATES Orchestrator Web Frontend

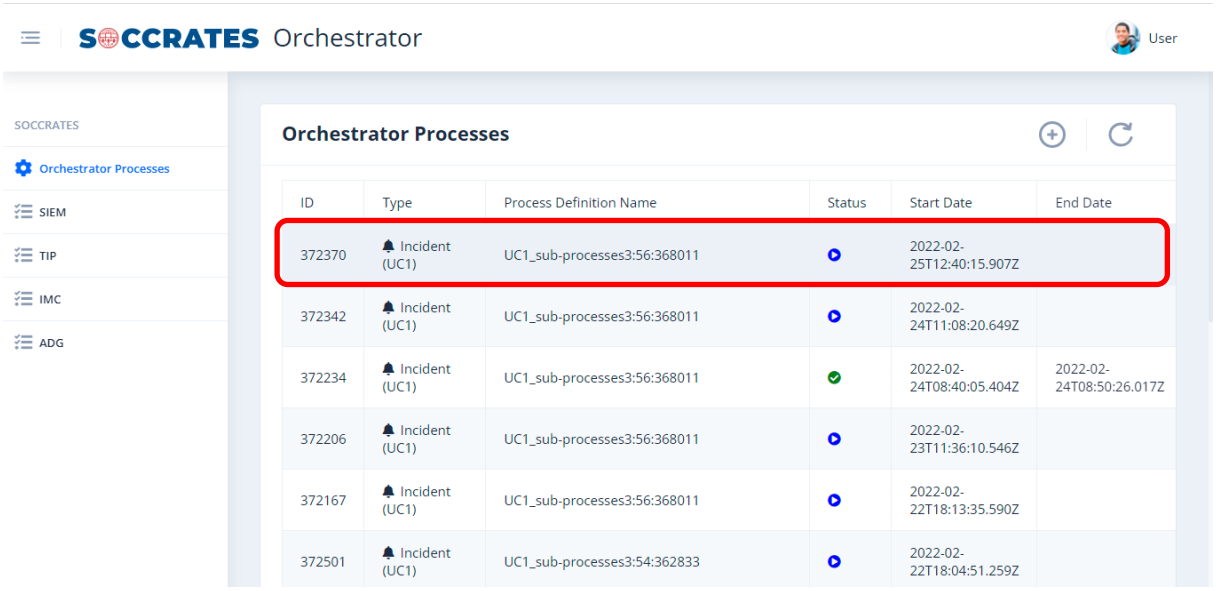


Figure 3-3 : New process registered in SOCCRATES Orchestrator with UC1 triggering

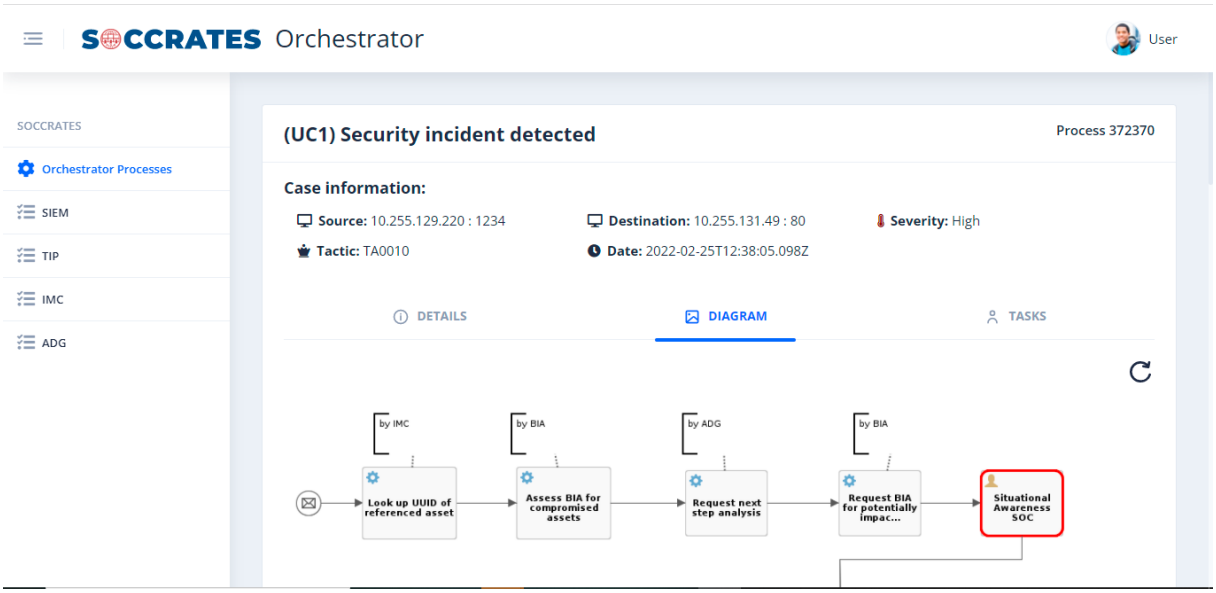


Figure 3-4 : UC1 Triggering message details in SOCCRATES Orchestrator Web Frontend

3.1.2 UC2: Response on Reception of New Cyber Threat Intelligence

The result of the reception of a new CTI, such as exploit code discovered for a vulnerability, through the TIP is that the Use Case 2 is triggered (see Figure 3-5). A new orchestrator process is created as shown in the SOCCRATES Web Frontend (see Figure 3-6). The same occurs with the reception of CTI on a new technique associated with a threat actor (see Figure 3-7). The details of the triggering messages are different in each test case and can be also visualized in the details of the processes as shown in Figure 3-8 and Figure 3-9.

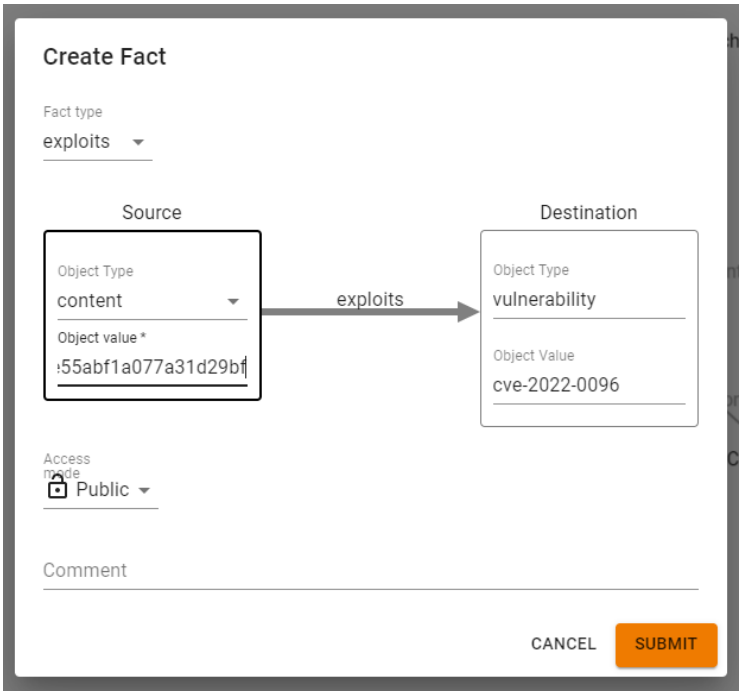


Figure 3-5 : UC2 Triggering in ACT (CTI on observed exploit for vulnerability)

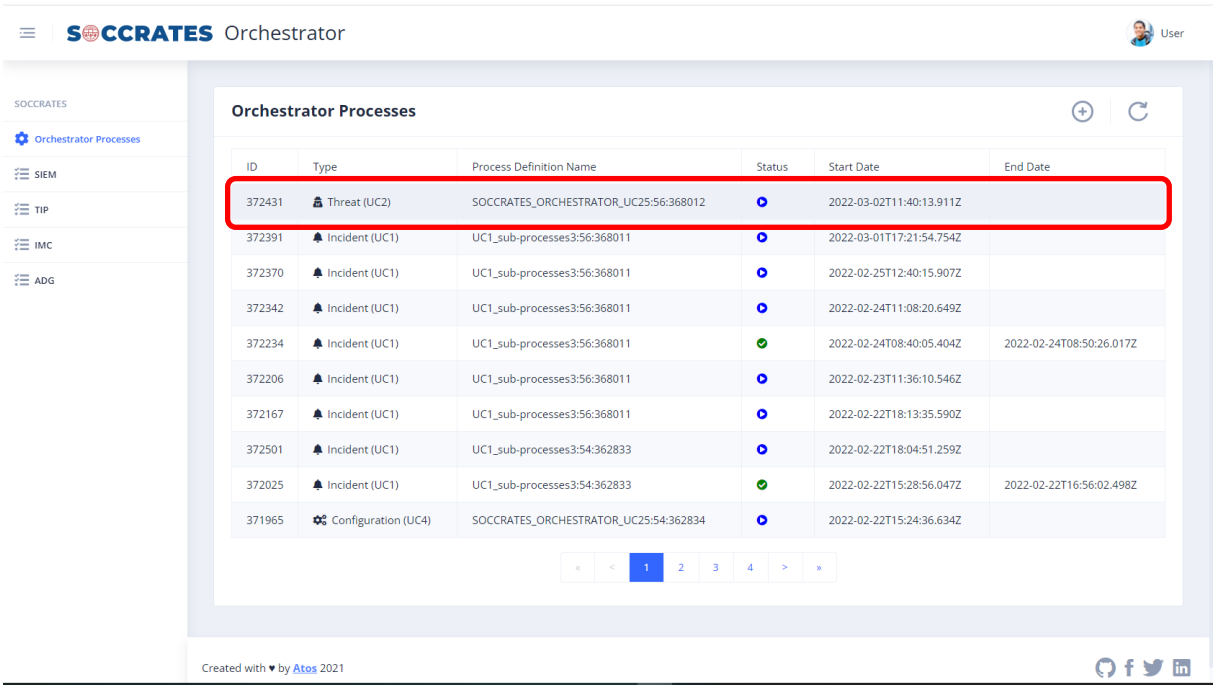


Figure 3-6 : New process registered in SOCCRATES Orchestrator with UC2 triggering (exploit for vulnerability)

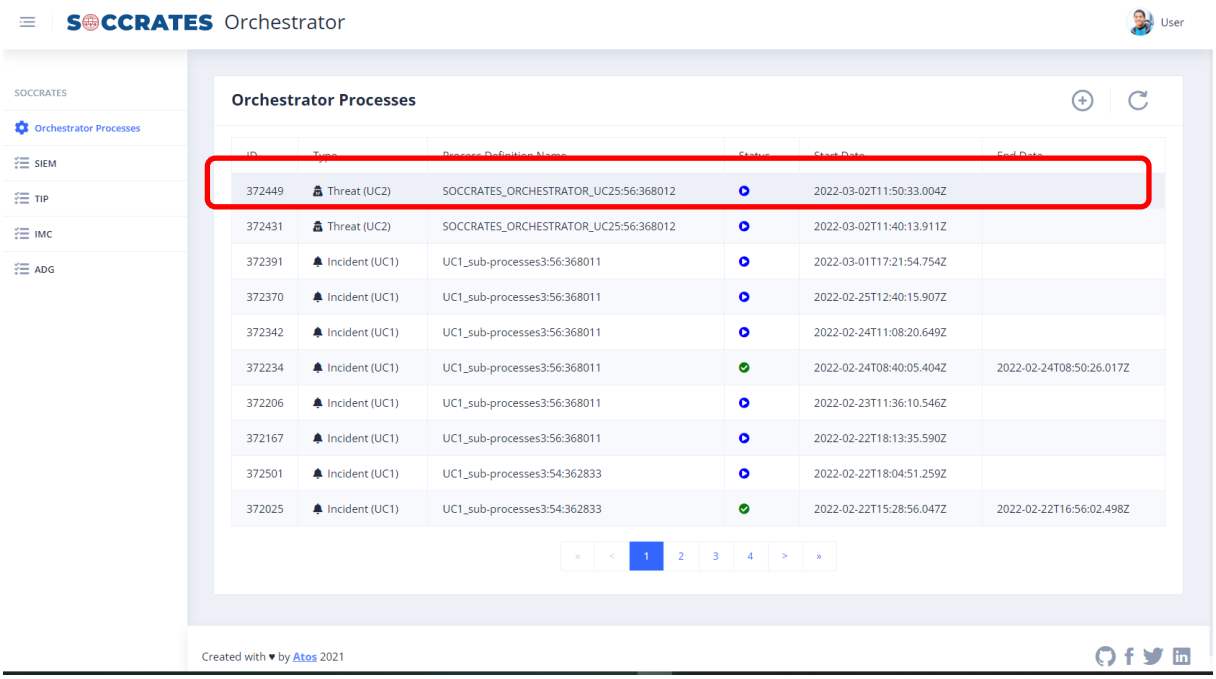


Figure 3-7 : New process registered in SOCCRATES Orchestrator with UC2 triggering (threat actor)

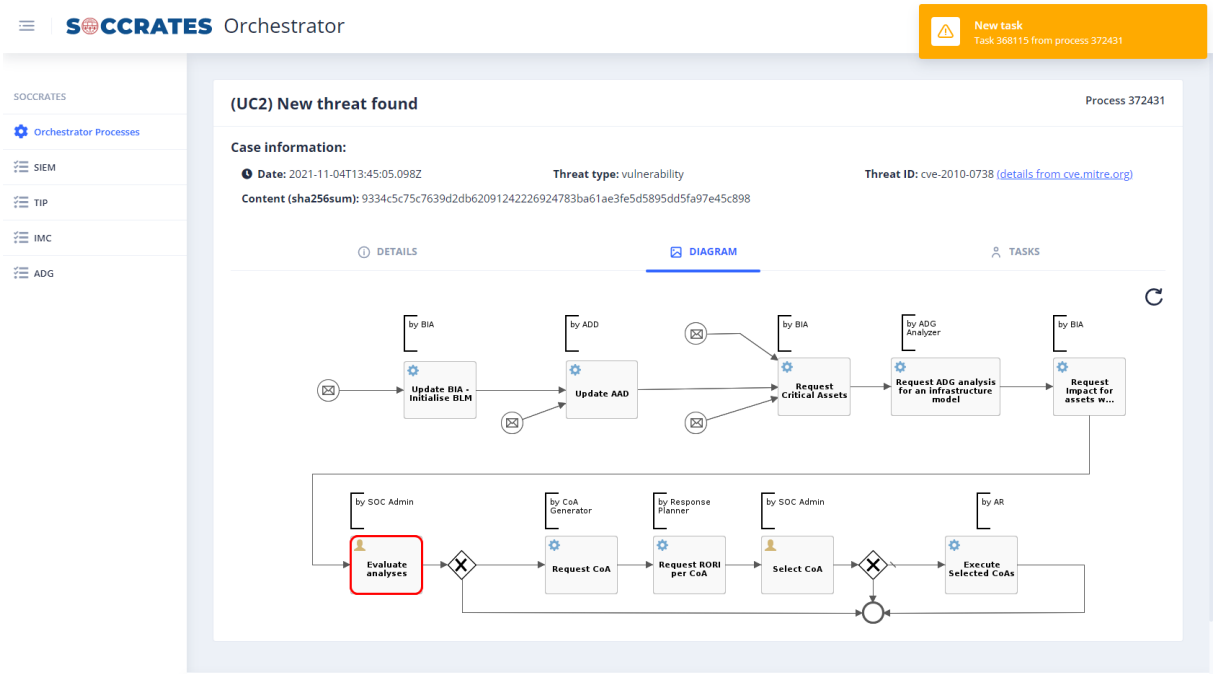


Figure 3-8 : Details about UC2 triggering message (exploit for vulnerability)

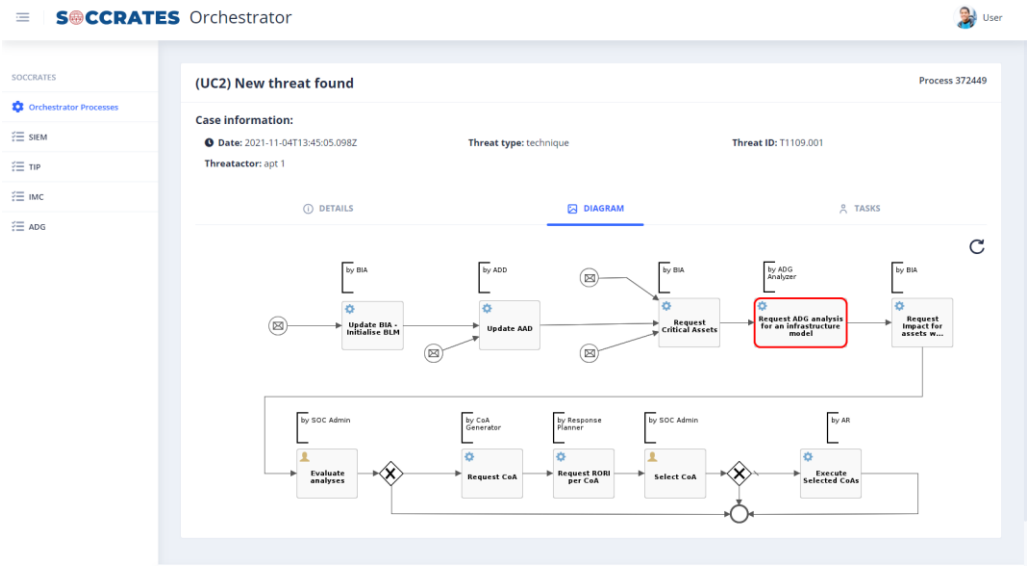


Figure 3-9 : Details about UC2 triggering message (new threat technique and threat actor)

3.1.3 UC3: Response on Discovery of New Vulnerable Assets

As it is described in the test case A.4 included in the Appendix A, to test the triggering of the Use Case 3 we have created in the IMC interface a new fact of type “affects” that connects a new vulnerability with a software (see Figure 3-10).

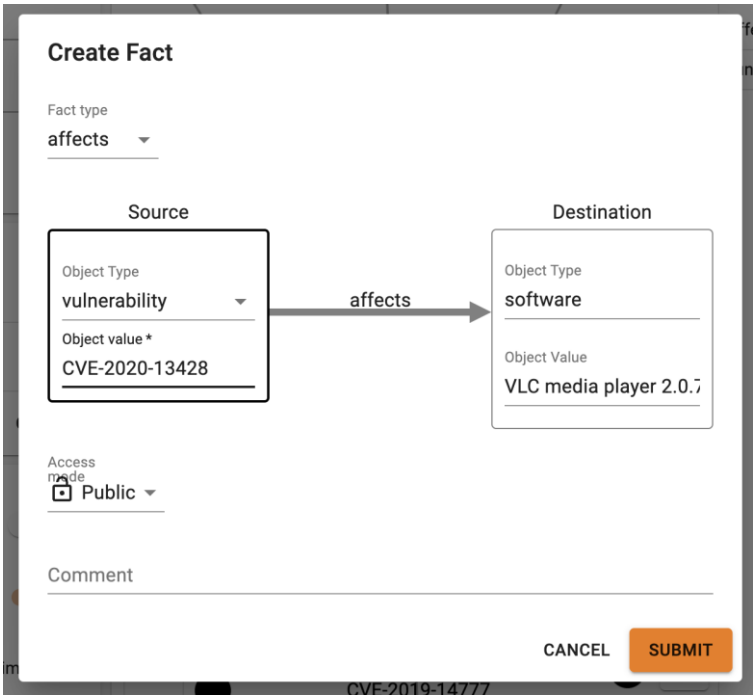


Figure 3-10 : UC3 Triggering (detection of new vulnerability)

The result is that a new orchestrator process is launched of type Vulnerability (UC3) as shown in Figure 3-11. The details of the new vulnerability detected included in the triggering message are shown in Figure 3-12.

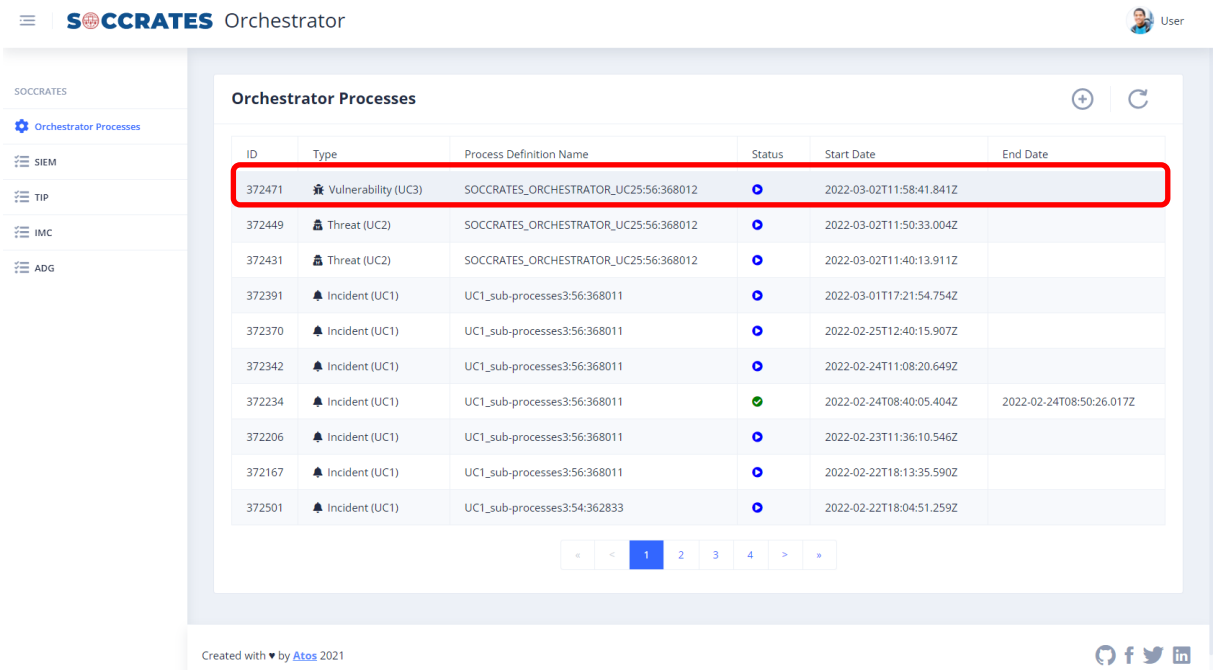


Figure 3-11 : New process registered in SOCCRATES Orchestrator with UC3 triggering

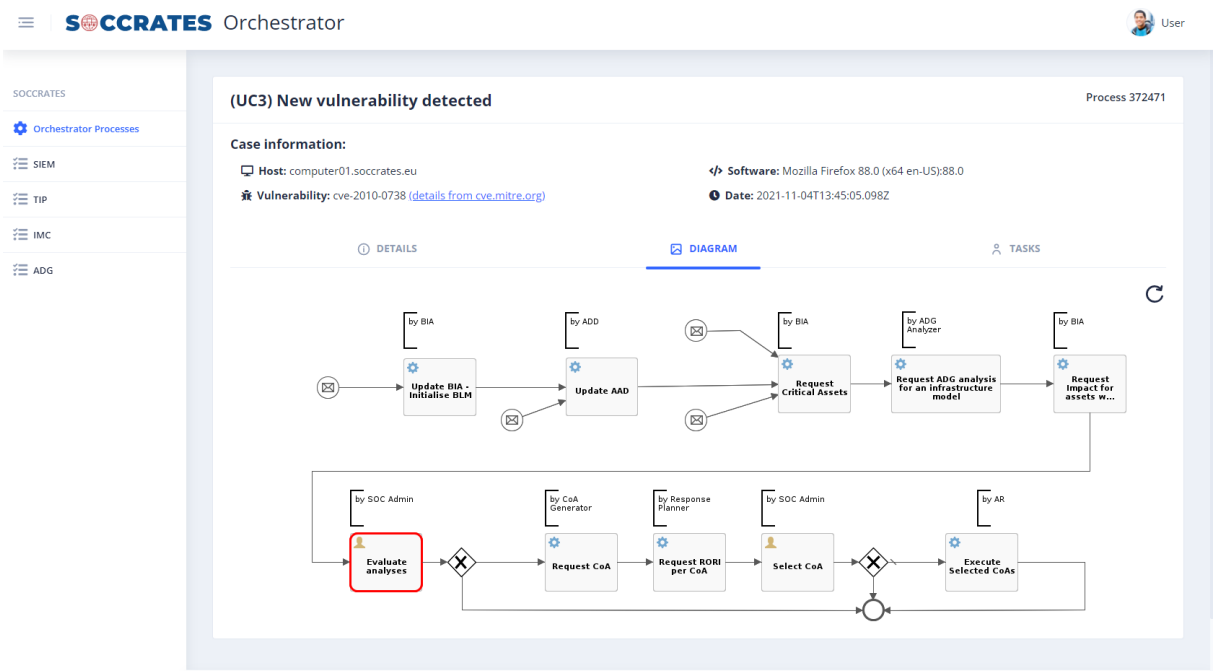


Figure 3-12 : Details of UC3 triggering message in Web Frontend

3.1.4 UC4: Response on Discovery of System Configuration Change

As defined in test case A.5 included in the Appendix A, to trigger the use case 4 it is necessary to create a rule associated to a ruleset through the IMC graphical interface as shown in Figure 3-13.

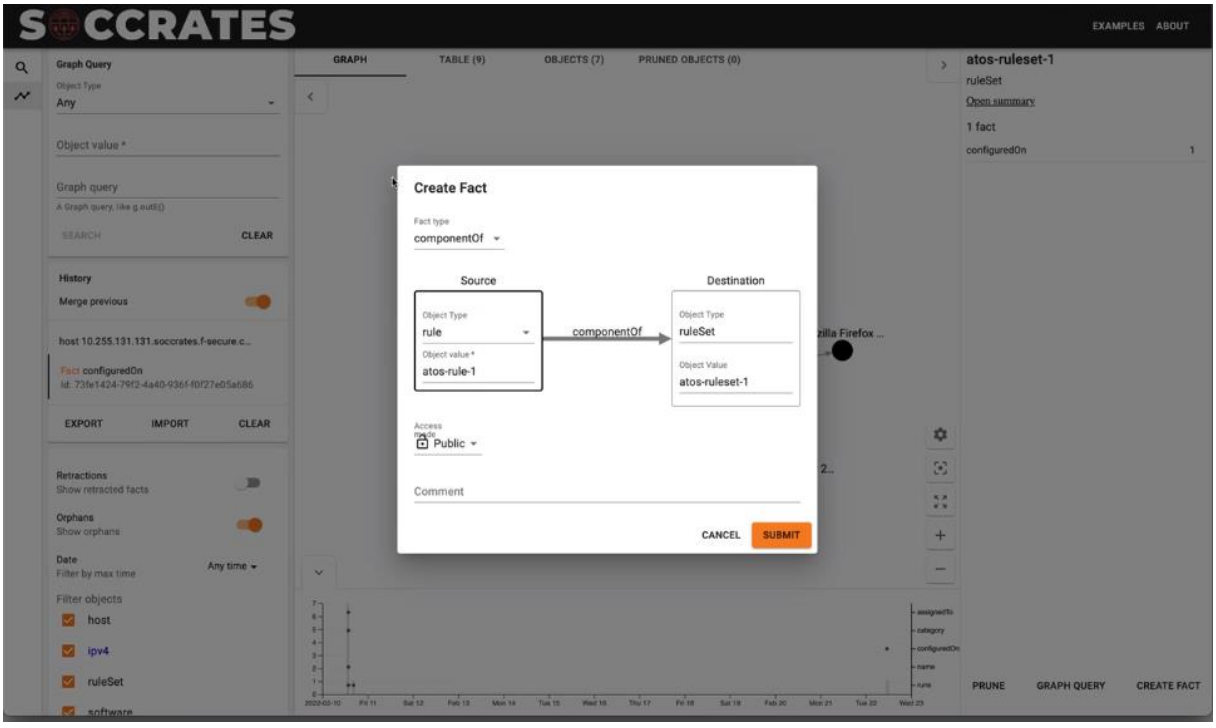


Figure 3-13 : UC4 Triggering (creation of configuration change)

The result is that a new orchestrator process is started and it appears in the SOCCRATES Orchestrator Web Frontend with the type “Configuration (UC4)” as shown in Figure 3-14. The details of the triggering message are shown in Figure 3-15.

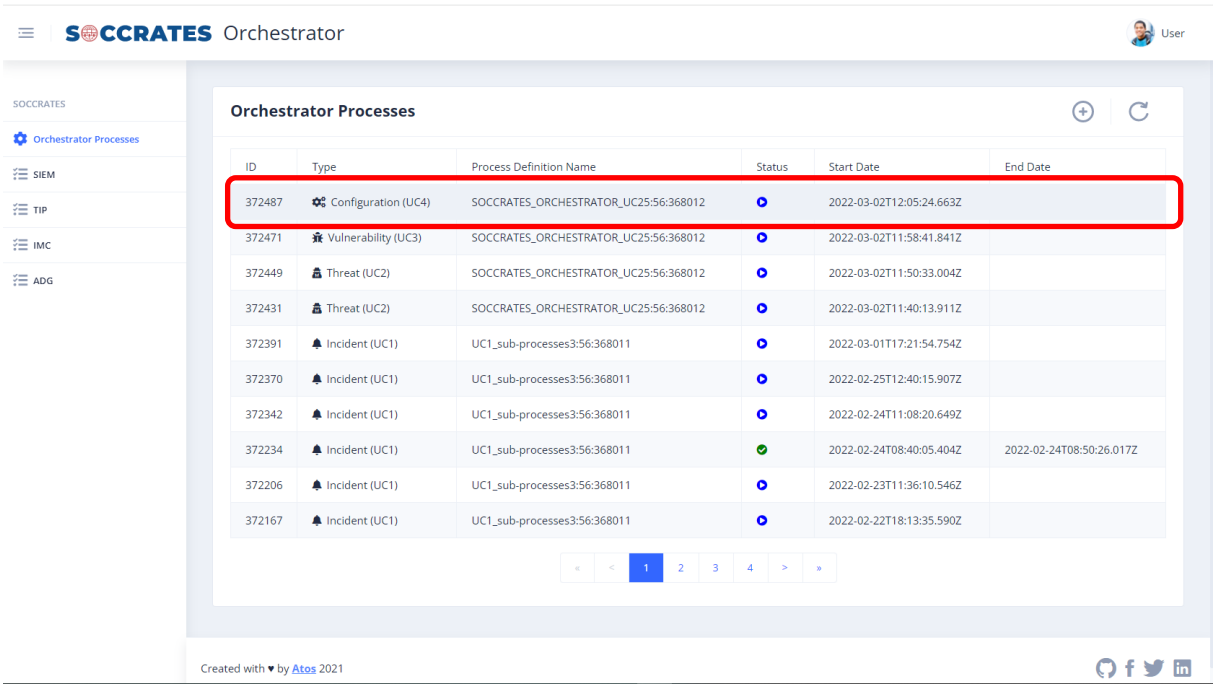


Figure 3-14 : New process registered in SOCCRATES Orchestrator with UC4 triggering

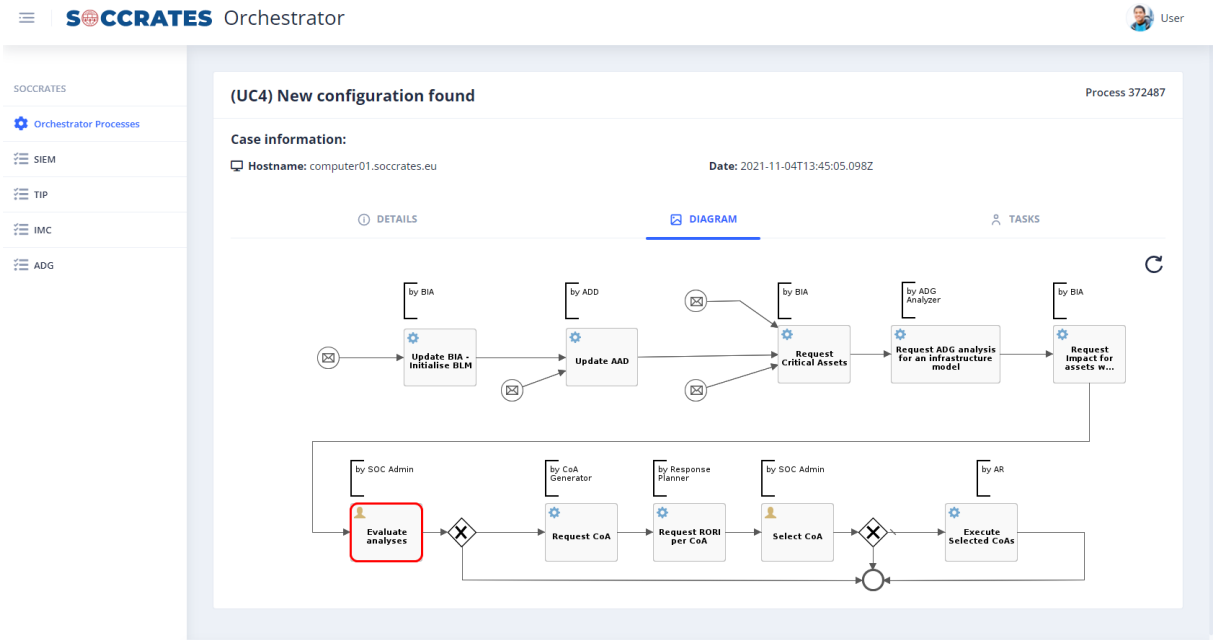


Figure 3-15 : Details of UC4 triggering message in Web Frontend

3.1.5 UC5: Response on Deployment of New Systems in Infrastructure

Use Case 5 is triggered, as defined in test case A.6 included in Appendix A, when a new system is added to the infrastructure as shown in Figure 3-16.

Figure 3-16 : UC5 triggering (creation of new host)

When a new IP address is assigned to the new host created, a triggering message is sent to the SOCCRATES Orchestrator and a new process is started in Activiti, as shown in the SOCCRATES Orchestrator Web Frontend (see Figure 3-17). The details of the triggering message can be also visualized in the frontend as shown in Figure 3-18.

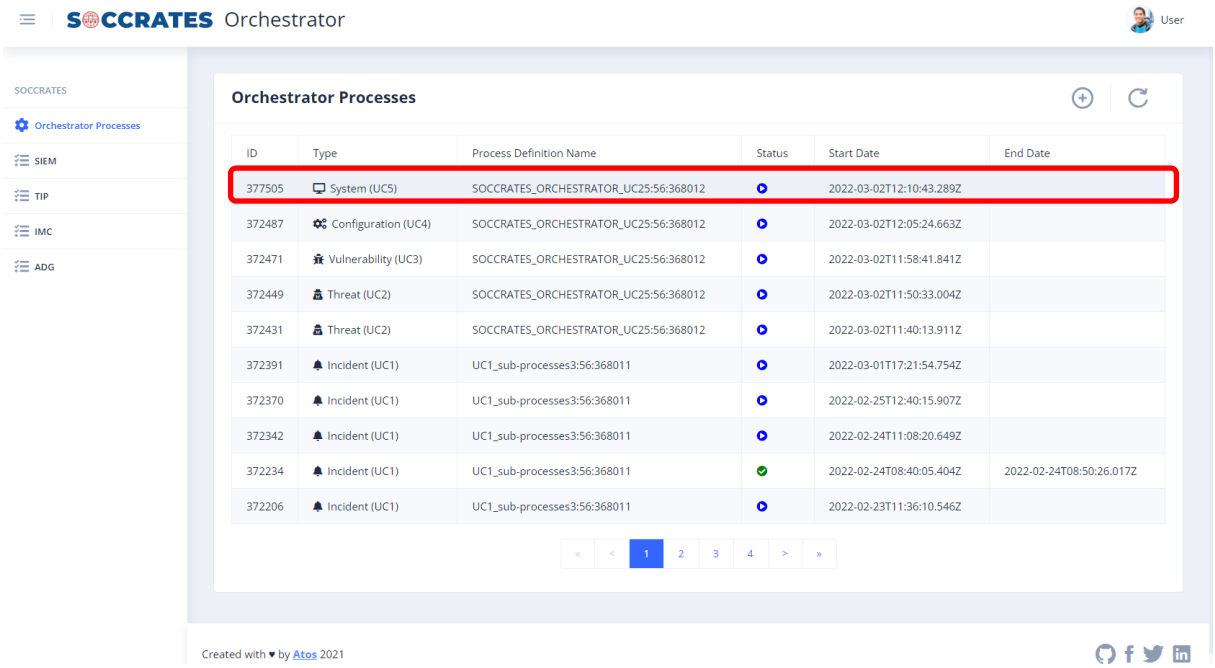


Figure 3-17 : New process registered in SOCCRATES Orchestrator UC5 Triggering

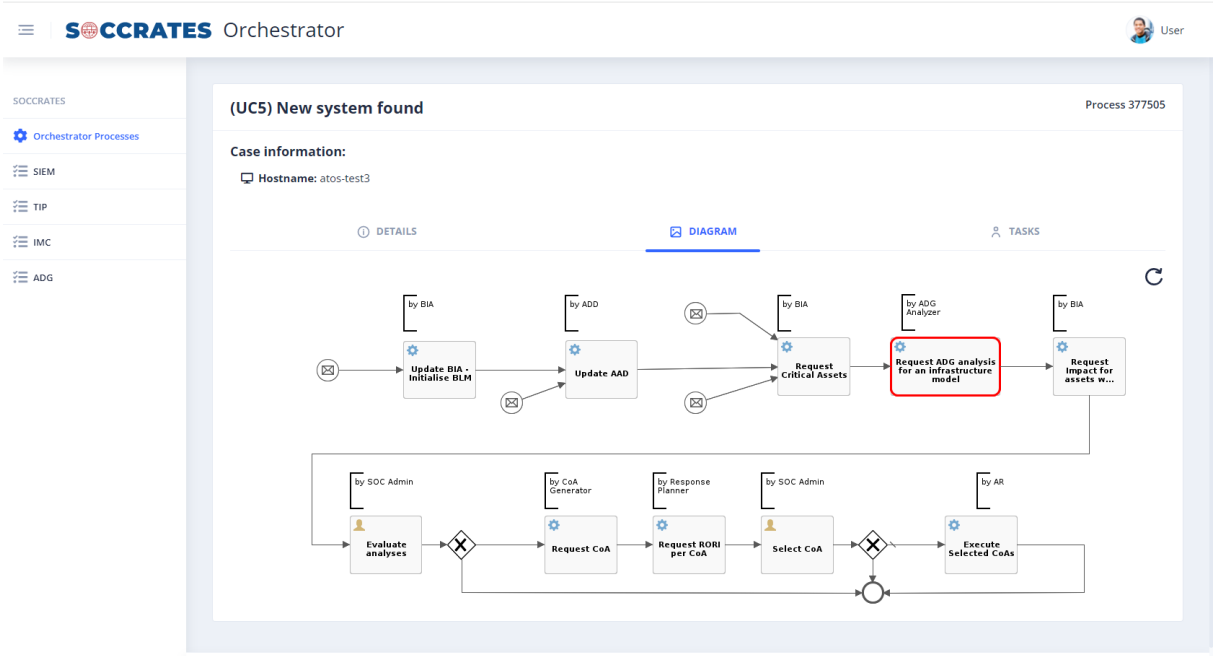


Figure 3-18 : Details of UC5 triggering message

3.2 Integration Activiti – Cortex Analysers / Responders

In this section, we report on integration part B in Figure 2-1. From the SOCCRATES Activiti Engine, the Orchestrator Core module is invoked to communicate with the SOCCRATES Cortex Analysers and Responders, using the implemented data models in the Orchestrator Core for each Cortex Analyser and Responder according to the definition provided in deliverable D2.3 [1].

We have defined the test cases B.1 to B.11 included in Appendix A to test the integration of the Orchestrator Core with the different Cortex Analysers and Responders.

For performing these tests, we have implemented a test suite in Java with JUnit which includes a different test for each Cortex Analyser and Responder that interacts with the Orchestrator during the execution of the use cases. In Figure 3-19 is shown an example of a test case. Each test case uses a fixed JSON ("testJson") for performing the request to the Cortex A/R.

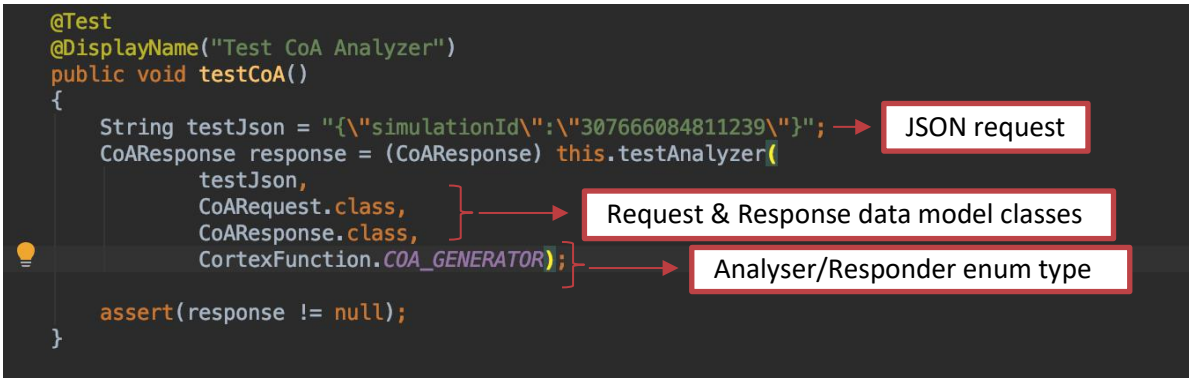


Figure 3-19 : Orchestrator Core JUnit test suite

Additionally, it uses the classes implemented to map the request and response of the Cortex A/R from JSON to java Objects. Finally, it also uses the "CortexFunction" enum, which provides the ID of

the analyser or responder that must be passed to Cortex. All those parameters are used to call to the function “testAnalyzer” or “testResponder”, depending on the case, which performs the request to the Cortex A/R and returns the retrieved data from Cortex. If we get a response from Cortex and the JSON received can be correctly mapped to the expected response class, then the test can be concluded as successful.

It is worth mentioning that all the classes, enums and functions used in these tests are also used in the Orchestrator Core, therefore we are able to validate its integration with Cortex by using the same implemented logic.

In Figure 3-20 is shown the result of executing this test suite, which includes the different tests for testing all the Cortex A&R.

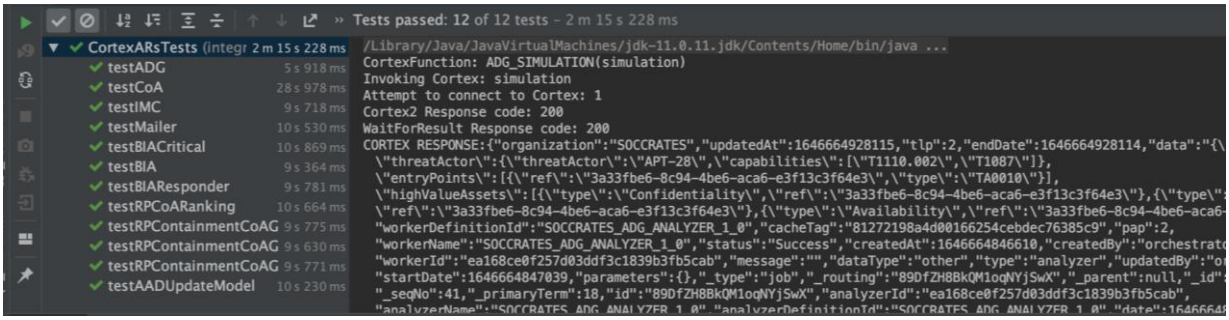


Figure 3-20 : Result of JUnit execution

The different tests shown in Figure 3-20 and the used Cortex Analyser / Responder are also shown in Table 3-1.

Table 3-1 : Integration Activiti with Cortex – Relationship between test names and Cortex A&R

Test name	Cortex Analyser / Responder
testADG	SOCCRATES_ADG_ANALYZER_1_0
testCoA	SOCCRATES_Coa_Generator_1_0
testIMC	SOCCRATES_IMC_ANALYZER_1_0
testMailer	Mailer_1_0
testBIACritical	SOCCRATES_Business_GetCritical_Analyzer_1_0
testBIA	SOCCRATES_Business_Impact_analyzer_1_0
testBIAResponder	SOCCRATES_Business_Impact_responder_1_0
testRPCoARanking	SOCCRATES_RP_COA_RANKING_ANALYZER_1_2
testRPContainmentCoAGenera-tor4Containment	SOCCRATES_RP_CONTAINMENT_COA_GENERATOR_1_0
testRPContainmentCoAGenera-tor4StoppingExfiltration	SOCCRATES_RP_CONTAINMENT_COA_GENERATOR_1_0
testAADUpdateModel	SOCCRATES_AAD_RESPONDER_1_0

3.3 Integration Cortex Analysers / Responders – Components

In this section, we report on integration part C in Figure 2-1. From the SOCCRATES Cortex Analyzers and Responders, the different components APIs are invoked to perform their functionalities. We have defined the test cases C.1 to C.14 included in Appendix A to test whether the integration of the Cortex analyzers/responders with the components has been correctly done.

Each of these test cases has been implemented in a function included in the Python script *test_cortex_analyzer.py* and they have been executed using Pytest[6]. An example of these functions is shown in Figure 3-21. Each function has been included in a class associated to the component tested and marked with a label to identify the component and if it is associated to an analyzer or responder. In this way, it can be more easily automate the execution of all the test cases or just a subset of them. The function *check_analyzer(analyzer)* invoked in each function sends a POST request to the Cortex API to run a specific analyzer or responder (e.g. http://10.0.2.3:9001/api/responders/9db22dd73574180f3b17402fcf3f5736/run) with the associated request data provided as an argument in each function. If the response of the Cortex job associated to the execution of the analyzer/responder includes a field report with "success=True", we can assume the integration is correct since the component has responded successfully to the request and generated the report with the response.

```
@pytest.mark.IMC
class TestIMC:
    @pytest.mark.analyzer
    def test_case_C1(self):
        print("Test Case C.1: IMC Cortex Analyzer")
        analyzer = {
            "name": "Test Case C.1: IMC Cortex Analyzer",
            "id": "96ad6dfabbb15db5540b05cf168468b0",
            "type": "analyzer",
            "dataType": "other",
            "data": "{\\\"ipv4\\\":\\\"10.255.131.163\\\"}"
        }

        assert check_analyzer(analyzer)
```

Figure 3-21 : Integration test function example

The result of the execution of all the integration tests is shown in Figure 3-22.

[6] https://docs.pytest.org/en/7.0.x/

```
root@soccrates:/home/xlsiem/orchestrator/test# pytest -v -s test_cortex_analyzers.py
============================= test session starts ==============================
platform linux -- Python 3.6.9, pytest-7.0.1, pluggy-1.0.0 -- /usr/bin/python3
cachedir: .pytest_cache
rootdir: /home/xlsiem/orchestrator/test, configfile: pytest.ini
collected 16 items

test_cortex_analyzers.py::TestIMC::test_case_C1 Test Case C.1: IMC Cortex Analyzer
PASSED
test_cortex_analyzers.py::TestADG::test_case_C2 Test Case C.2: ADG Cortex Analyzer (with entry points)
PASSED
test_cortex_analyzers.py::TestADG::test_case_C3 Test Case C.3: ADG Cortex Analyzer (with assets of interest)
PASSED
test_cortex_analyzers.py::TestADG::test_case_C4 Test Case C.4: ADG Cortex Analyzer (with vulnerabilities)
PASSED
test_cortex_analyzers.py::TestADG::test_case_C5 Test Case C.5: ADG Cortex Analyzer (with threat actor)
PASSED
test_cortex_analyzers.py::TestBIA::test_case_C6 Test Case C.6: BIA Cortex Analyzer (impact analysis)
PASSED
test_cortex_analyzers.py::TestBIA::test_case_C7 Test Case C.7: BIA Cortex Analyzer (get critical)
PASSED
test_cortex_analyzers.py::TestBIA::test_case_C15 Test Case C.15: BIA Cortex Responder
PASSED
test_cortex_analyzers.py::TestRP::test_case_C8 Test Case C.8: RP Coa Ranking Cortex Analyzer (no containment)
PASSED
test_cortex_analyzers.py::TestRP::test_case_C9 Test Case C.9: RP Coa Ranking Cortex Analyzer (containment)
PASSED
test_cortex_analyzers.py::TestRP::test_case_C10 Test Case C.10: RP Containment Coa Generator Cortex Analyzer (stop_exfiltration)
PASSED
test_cortex_analyzers.py::TestRP::test_case_C11 Test Case C.11: RP Containment Coa Generator Cortex Analyzer (containment)
PASSED
test_cortex_analyzers.py::TestCoA::test_case_C12 Test Case C.12: CoA Generator Cortex Analyzer
PASSED
test_cortex_analyzers.py::TestADD::test_case_C13 Test Case C.13: AAD Cortex Analyzer (new asset)
PASSED
test_cortex_analyzers.py::TestADD::test_case_C14 Test Case C.14: AAD Cortex Analyzer (new vulnerability)
PASSED
test_cortex_analyzers.py::TestAR::test_case_C16 Test Case C.16: AR Cortex Analyzer
PASSED

============================== 16 passed in 385.32s (0:06:25) ==============================
```

Figure 3-22 : Result pytest integration testing

In the following subsections we include in detail the test cases associated to each analyzer/responder including the request data sent and the response provided by each of the Cortex analyzers and responders as it is shown in the Cortex GUI.

3.3.1 IMC Cortex Analyzer

The IMC Cortex Analyzer is invoked from Activiti in the Use Case 1 to get the UUID reference for a compromised asset or for collateral affected IP address. In both cases, the IP address is only available, whereas the different tasks included in the use case workflow need to include in the request data the asset reference. In the first case, once the UC1 triggering message has been received, the request data is prepared for the IMC Analyzer with the IP of the destination IP in the SIEM message (in case of an initial access – event TA001), the source IP (in case of an exfiltration – event TA0010) or both, source and destination IPs (in case of a lateral movement - event TA008). In the second case, the analyzer will be invoked for each of the collateral affected IPs introduced by the analyzer through the SOCCRATES Web Frontend.

To test the integration of this Cortex Analyzer with the IMC component, test case C.1 has been executed, which is described in Appendix A.

3.3.2 ADG Cortex Analyser

The ADG Cortex Analyzer is invoked from Activiti in all the use cases workflows, but depending on the use case the request data will be different. Entry points are optional and only included in use cases 1 (for compromise assets with the information provided by the SIEM in the triggering message) and 2 (in case they came from TIP). Assets of interest are only included in the use cases 4 and 5 in case the triggering message provides some reference asset. Vulnerabilities are only included in the use case 3 when a new vulnerability is detected. Threat actor information can be provided in the case of the use case if it is provided by the TIP. Consequently, we have defined different test cases to cover all these possibilities of optional fields.

To test the integration of this Cortex Analyser with the ADG Analyser component, test cases C.2, C.3, C.4 and C.5 have been executed, which are included in Appendix A.

This analyzer interacts not only with the Attack Defence Graph analyzer but also with the TIP and IMC. These interactions are included in the test cases listed above. The ADG Cortex analyzer interacts with the IMC in all test cases and with the TIP in test case C.5.

3.3.3 BIA Cortex Analyser

There are two different BIA Cortex Analysers interacting with the Business Impact service: one to request the impact probability on the most important business functions and another to request the criticality of the assets. Test cases C.6 and C.7 included in the Appendix A have been executed to test the integration of each of these Cortex Analyzers with the BIA component. A correct output from the Cortex analyzer means the model was successfully built and consequently the integration of the analyzer with the BIA component and between the BIA and the IMC (to collect the list of nodes in the infrastructure and therefore build the model) is correct.

3.3.4 RP Cortex Analyser

Two different Cortex Analysers are used to interact with the Response Planner component in the Use Case 1 workflow: the Ranking CoA Analyzer and the Containment CoA Generator. In the RP Ranking CoA service, depending on the input containment parameter, the RP will compute a different metric. When containment=true in input, the inputs CoAs should come from the Containment CoA Generator, the RP will interact with the BIA and compute a Business Impact for each CoA. When containment=false, the input CoAs should come from the CoA Generator, and the RP will compute a RORI.
To test the integration of the Ranking CoA Cortex Analyser with the Response Planner component in those different situations, test cases C.8 and C.9 have been executed.
The request data for the Containment CoA Generator Cortex Analyzer makes use of the source and destination IP addresses included in the SIEM triggering message and the option selected by the SOC analyst through the SOCCRATES Web FrontEnd. But the behaviour of the component will depend on the situation where this analyser is invoked, in the preliminary response phase and in the final containment phase. Consequently, we have defined two different test cases with a different value in the parameter type (“stop_exfiltration” or “containment”), C.10 and C.11, to test the integration of this Cortex Analyser with the Response Planner component in both situations.

3.3.5 COA Cortex Analyser

The CoA Cortex Analyser invokes the Course of Action Generator to return a set of defences that are suggested to be activated in the infrastructure, based on the information provided by a previous ADG analysis, whose simulation id is included in the request data.

To test the integration of this Cortex Analyzer with the Course of Action Generator component, test case C.12 has been executed, included in the Appendix A.

3.3.6 AAD Cortex Responder

The AAD Cortex Responder interacts with the AI based Attack Detection to update the model with a new asset or a new vulnerability. To test the integration of this Cortex Responder with the AI based Attack Detection component, two different test cases, C.13 and C.14 included in the Appendix A, have been defined and executed.

3.3.7 BIA Cortex Responder

The BIA Cortex Responder invokes the Business Impact Analyser to request the model is updated with a new asset added to the infrastructure. To test the integration of this Cortex Responder with the BIA component, test case C.15 included in the Appendix A has been defined and executed.

3.3.8 AR Cortex Responder

Currently, it is only available in the SOCCRATES Platform the Automated Reconfiguration Email Cortex Responder which interacts with a SMTP Server to send an email with a summary of the course of actions selected by the SOC analyst. To test the integration of this Cortex Responder with the SMTP server, test case C.16 included in the Appendix A has been defined and executed.

3.4 Integration with SHS

The SOCCRATES TIP uses the SHS Threat Data Collection & Threat Prediction API, specified in D2.3 [1], to enrich newly received CTI. The TIP directly accesses the SHS API through a worker, the integration does not involve any other SOCCRATES components. To test the integration of the TIP with the SHS Threat Data Collection & Threat Prediction API, test case C.17 included in Appendix A has been defined and executed. To test the integration of the AAD (detection component *middleman* with the SHS Threat Data Collection & Threat Prediction API, test case C.18 included in Appendix A has been defined and executed.

3.5 Web Front End

The testing of the Web Front End covers the integration of the different components that are directly involved with it. In Figure 3-23 are depicted the two components that form the Orchestrator UI: the Frontend and the Backend, both running in different Docker containers.

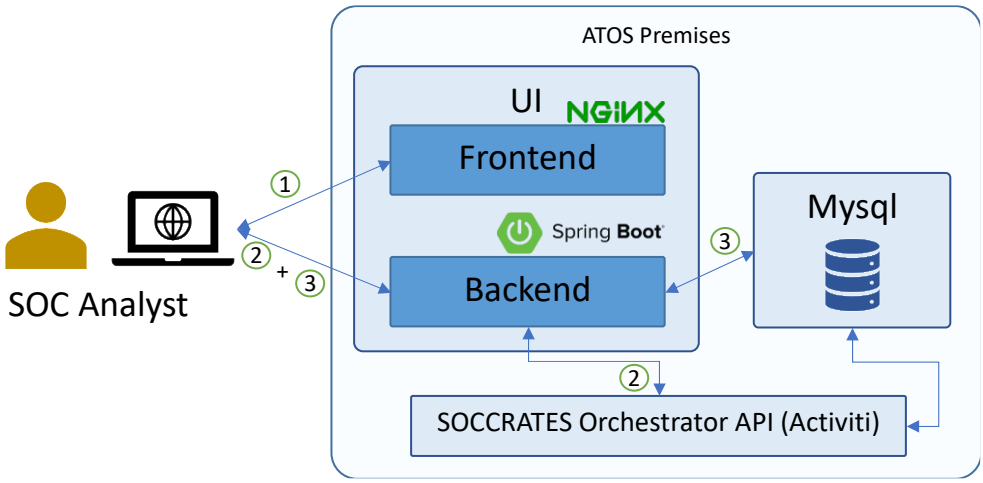


Figure 3-23 - SOCCRATES UI components diagram

The Frontend provides the graphical interface resources that are used by the user browser for displaying the Web UI, like the HTML code, the CSS styles and the executed Javascript code. It is worth mentioning that the Frontend code is executed in the user browser once downloaded, so every further interaction between the Frontend and the Backend is done from the user browser.

The Backend, instead, provides an API for the Frontend, which is used to access to the Activiti API and to access to the Orchestrator DB (MySQL) directly. This Backend API is accessed directly from the user browser through the Javascript code obtained from the Frontend, as depicted in Figure 3-23. The direct access to MySQL is used for logging into DB the user activity, like when the user receives a notification for a new user task that has to be completed, or when the user displays or completes this user task. Those logs are used to measure the real user time response.

More details about the Orchestrator UI components are provided in deliverable D6.4.

For testing the Orchestrator UI, the *Integration testing environment* described in section 2.3.1 has been used. The test consists of three phases with different checks, as shown in Figure 3-23 with green circles. The different testing phases are described below.

3.5.1 Test phase 1 – Get frontend sources

First, we access to the private Web URL http://10.0.2.3:4200/ (accessing through the ATOS VPN connection) and check that the Frontend resources are correctly loaded (CSS, HTML and JS files) from the Nginx web server. This test covers the interaction between the user and the Frontend component, when the sources are correctly downloaded. As shown in Figure 3-24, the main view is correctly loaded, where the HTML code with its styles are shown.

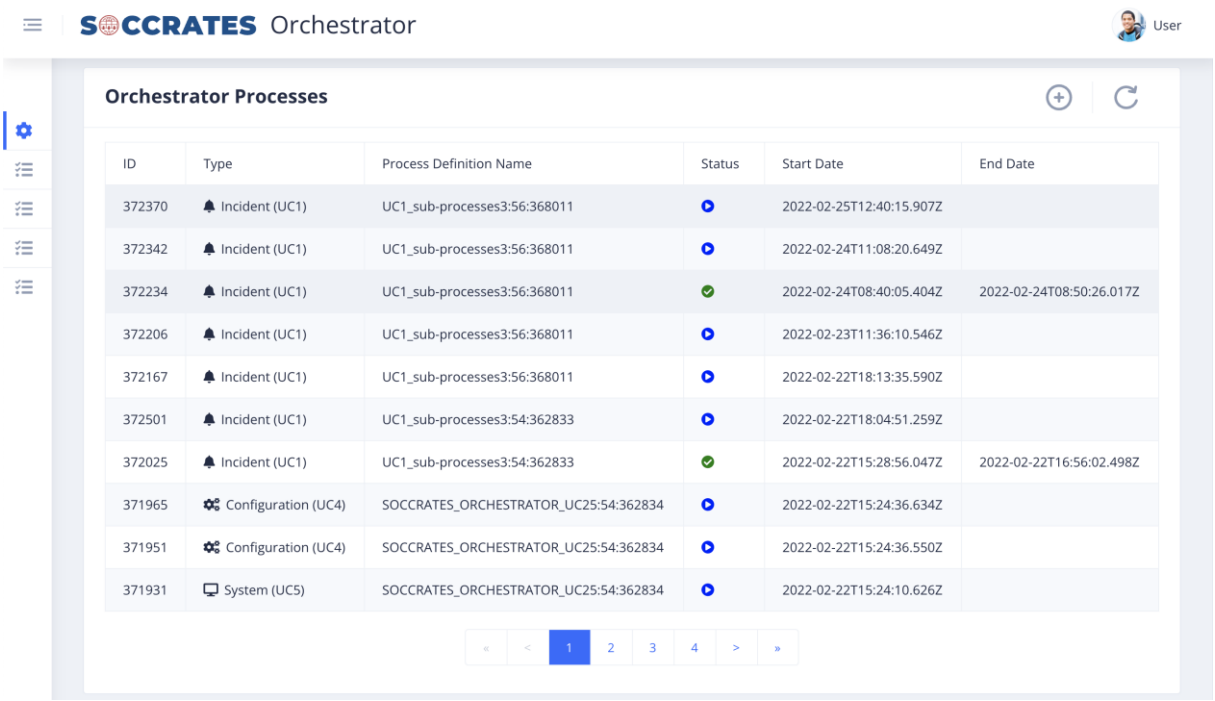


Figure 3-24 : Web Frontend correctly loaded

We also check that the sources have correctly been downloaded by using the browser inspector, as shown in Figure 3-25, where all the different sources have status “200”:

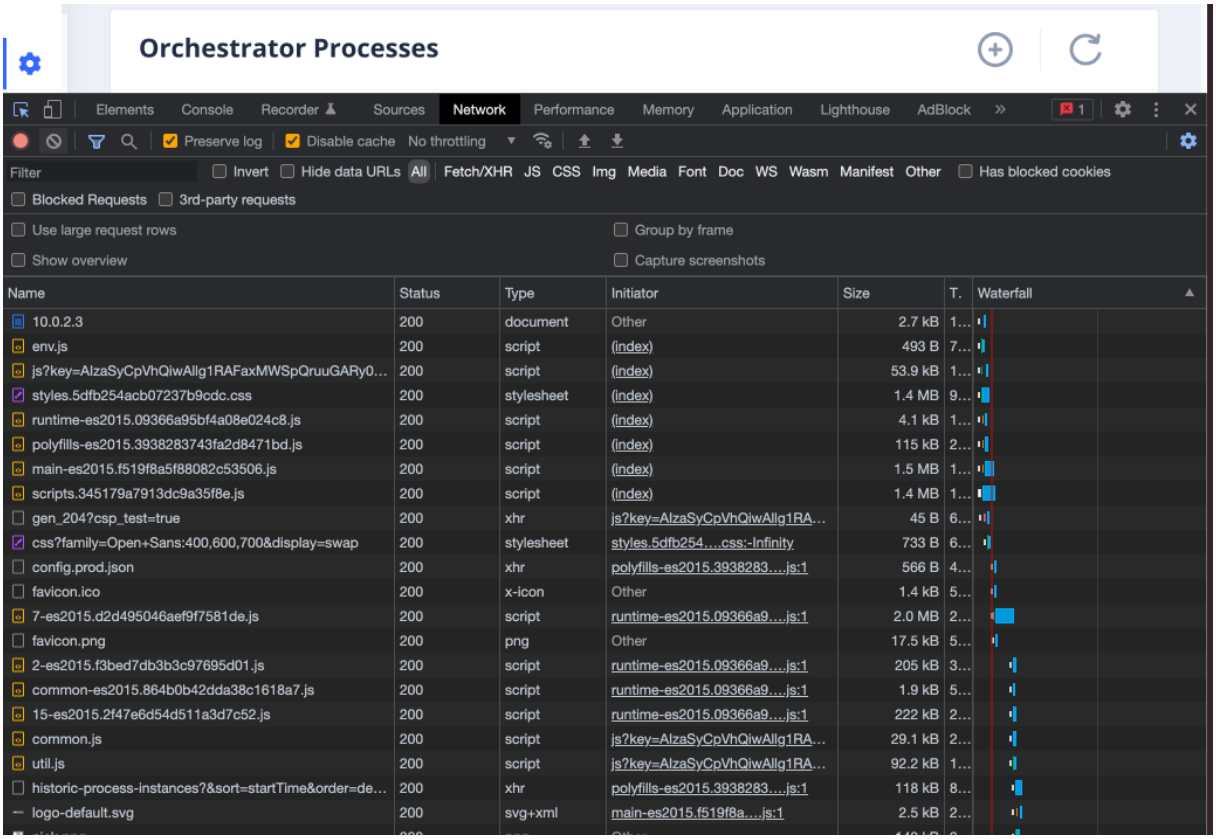


Figure 3-25 : Web Frontend sources correctly loaded

3.5.2 Test phase 2 – Get data from Activiti through backend

Afterwards, we check that the data has been loaded from the Backend API, the list of Orchestrator Processes. We can check it by simply looking at the main view or alternatively we can verify it using the browser inspector at the "Network" tab. This validates the integration of the Frontend code (running in the browser) with the Backend at the same time that the integration of the Backend with the SOCCRATES Orchestrator API. Figure 3-26 shows the data returned by the XHR request that loads the list of processes running in the Orchestrator.

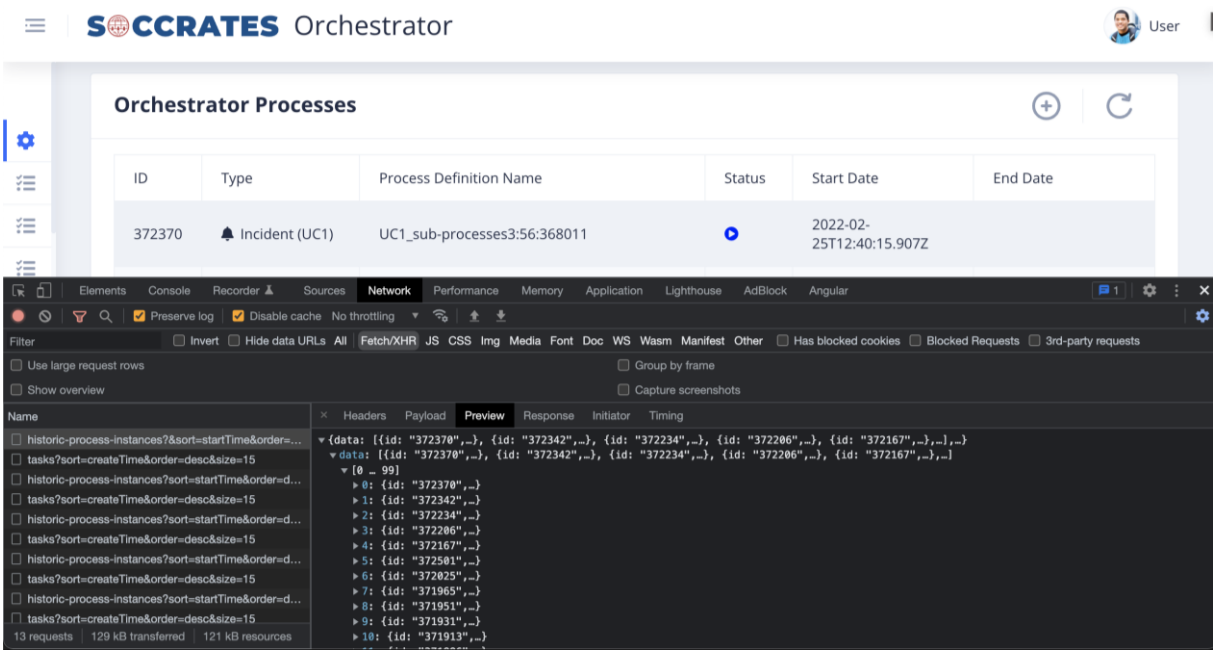


Figure 3-26 : Web Frontend loading data from Activiti through backend

3.5.3 Test phase 3 – Save data to DB through backend

Finally, we create the required conditions in the interface for saving data to the Orchestrator DB using the Backend. For that purpose, we need to generate new user tasks by triggering a new use case. First, we trigger a random UC from the interface to force the Orchestrator to create new user tasks and get the notification in the browser. Figure 3-27 shows the result of generating a new UC1 through the interface, as described in section 3.1.1.

Figure 3-27 : Starting a new UC from Web Frontend

Once the UC has started, we wait until we get the notification of a “New task”. The interface automatically logs this activity to the DB, so a request is done to the backend with this event information. Figure 3-28 shows how the notification is shown in the interface.

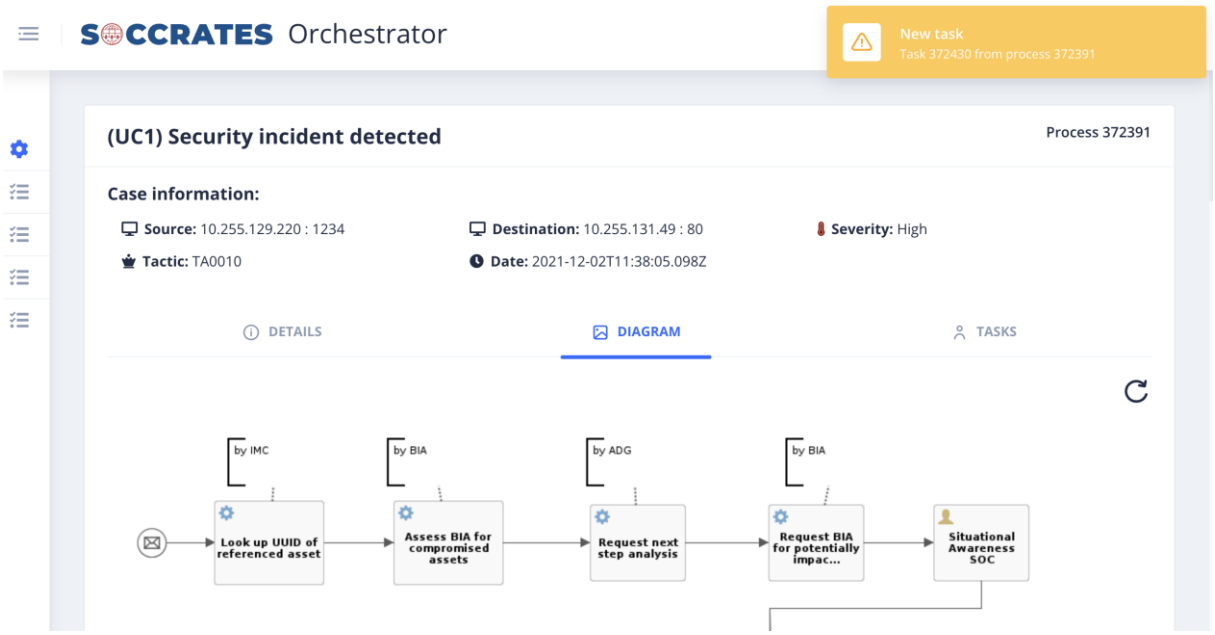


Figure 3-28 : Web Frontend - New task notification

We check that the request to the Backend to log this activity has been correctly performed to validate again the communication between the Frontend code (running in the browser) and the Backend, in addition to check that the log have correctly been stored in the Orchestrator DB. Figure 3-29 shows the overview of the request which has a status code of 200, which means it was correctly saved:

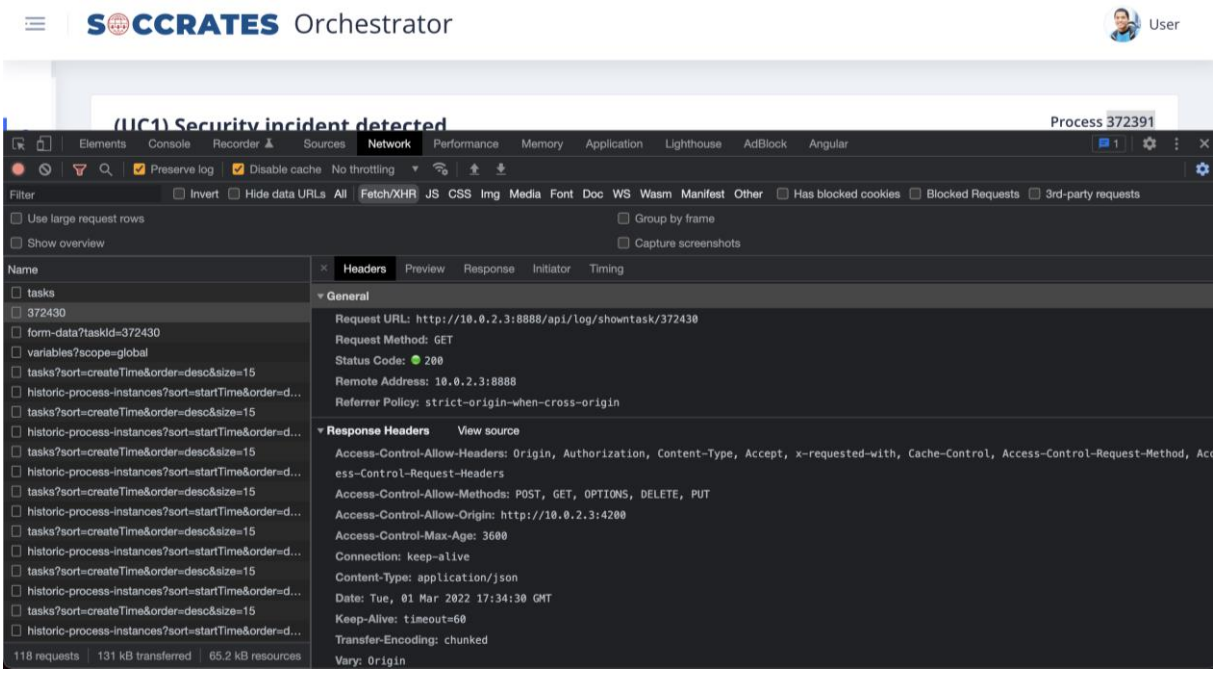


Figure 3-29 : Web Frontend - Request overview for logging user activity to DB

Figure 3-30 shows the response from the backend to this request, which returns the information regarding the created log entry:

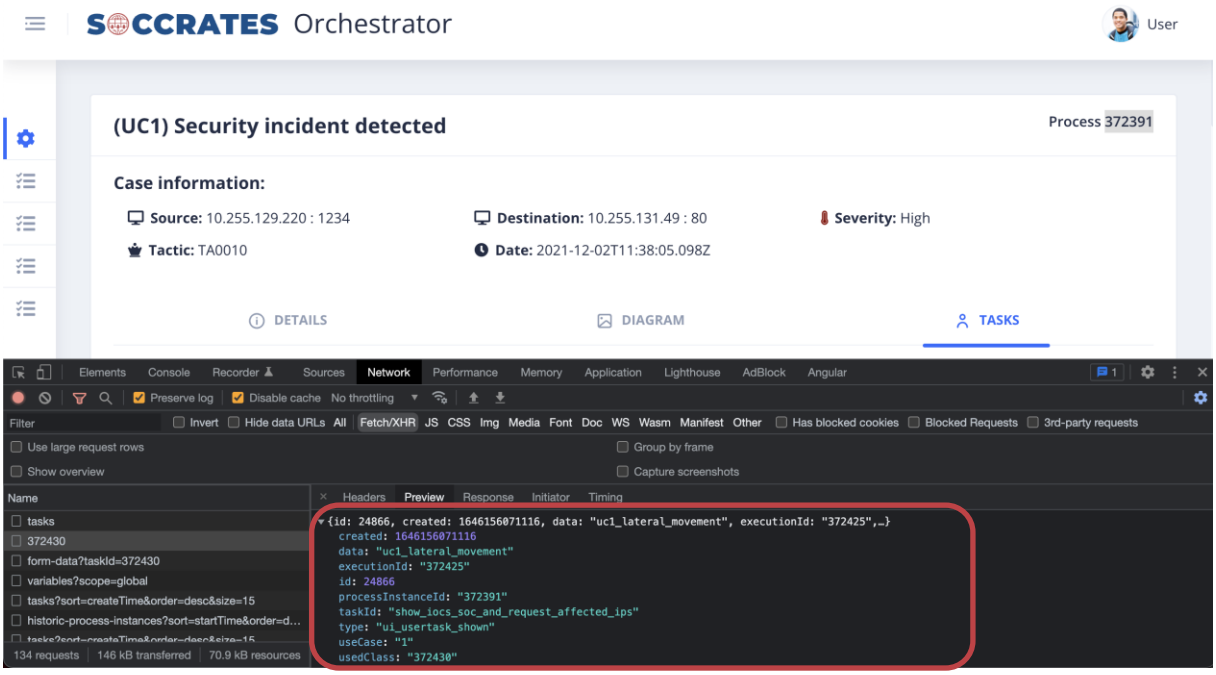


Figure 3-30 : Web Frontend - Logging user activity result

Additionally, we check the saved information in the DB table. Figure 3-31 shows the view of the table « orchestrator_log » that stores this information. As we can see, it has been correctly stored, so the test can be concluded as successful.

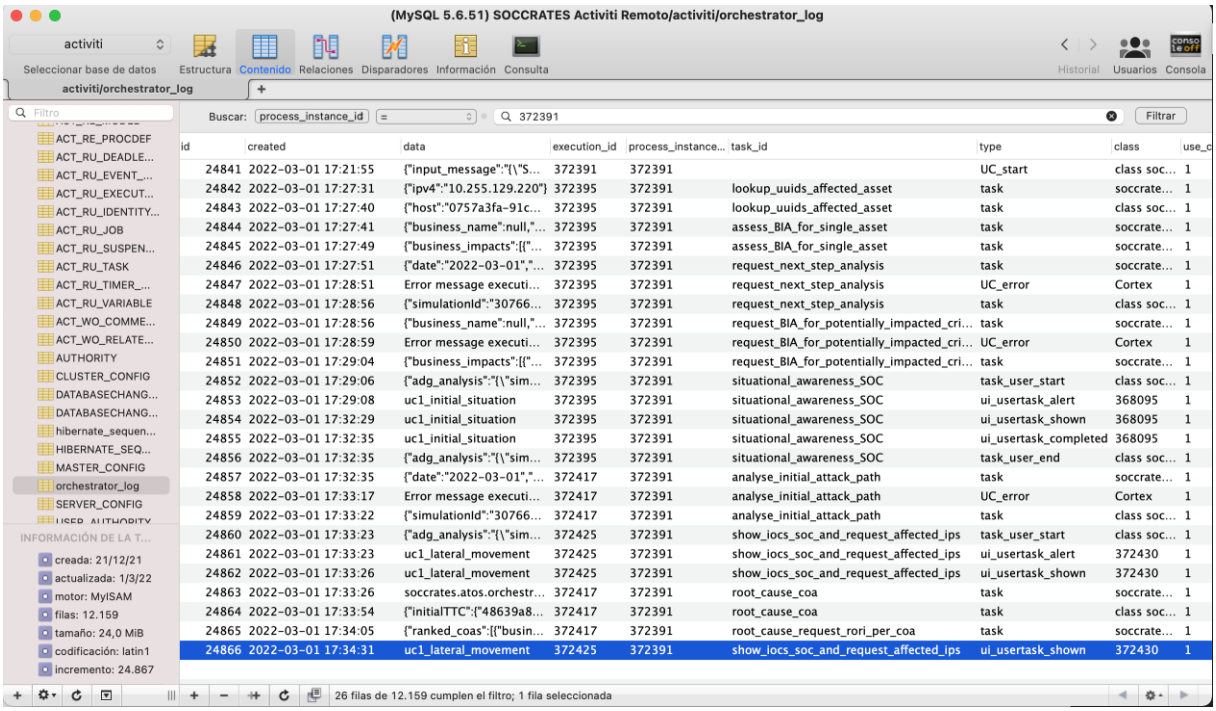


Figure 3-31 : Web Frontend - DB view of logged user activity

4 Platform Deployment Testing

4.1 Making the deployment declarative

The goal of platform deployment testing is to guarantee that the various components that make the SOCCRATES platform deploy without bugs. It's important to stress that in this activity we don't test the integration between components nor if the component works as intended. Instead, we focus on guaranteeing that the software will deploy.

In the first pilot, we deployed the AAD through Docker Compose, which is an imperative deployment technology. In imperative deployment technologies a machine is instructed to execute a specific action or a specific sequence of actions. Python, C or bash scripts are other examples of imperative scripting languages. The main issue with the imperative paradigm is that it is not strictly reproduceable. The same script may deploy on different machines in slightly different ways, introducing bugs that are hard to discover and triage. Furthermore, imperative deployments are less resilient as there are no default mechanisms to restore the state of an application after a simple system restart. These mechanisms have to be manually implemented.

We thus decided to migrate from Docker Compose to Docker Swarm. Swarm is a declarative deployment technology (Kubernetes is another example). In declarative paradigms we provide the machine with a desired state and we let the machine execute the commands required to reach the state. The machine monitors the deployment and takes appropriate action in case the actual state starts to differ from the desired state. For example, a swarm cluster made of multiple machines will reschedule workloads from one machine to another, if the former is turned off. Critically, declarative deployments are reproducible (if the desired state configuration follows best practices). We opted for Docker Swarm because it allows us to reuse the existing Docker Compose manifests, with only minimal changes required. Having made software deployment reproduceable, we then established a semi-automated methodology to test it.

4.2 Continuous Integration and Continuous Deployment (CI/CD)

We perform deployment testing through continuous integration and continuous deployment with Gitlab, the platform we use to store SOCCRATES software, see Figure 4-1. Every time a developer pushes a change to a file in one of our repositories, a Continuous Integration pipeline of tests is triggered. Among these tests, one checks for the validity of the Docker Compose manifest through the *docker-compose config* command[7]. Nothing is deployed by this command. If the manifest is invalid, the pipeline fails, and changes are not allowed to be merged into the main branch of the project. This simple, static, test helps prevents most bugs, which are easy to detect and prevent. This also moves error discovery from the operator (on the metaphorical "right") to the "left" towards the developer, which is the real expert of the application.

If the validation test is passed the Docker Swarm stack is deployed automatically in the TNO staging environment, which is described in more detail at section 2.3.2. This provides a dynamic validation of the properties that are not checked by the previous test, for example, by checking the actual availa-

[7] https://docs.docker.com/compose/reference/config/

bility of the docker images, secrets and configurations described in the manifest. This includes vulnerability scanning of the containers using Grype[8], see Figure 4-3. This database is updated at the beginning of each scan, but an update can also be triggered manually. Again, if this step fails changes are not allowed to be merged to the project main branch. On the other hand, if this step completes successfully, one can be confident that the component will deploy at the pilot sites too.

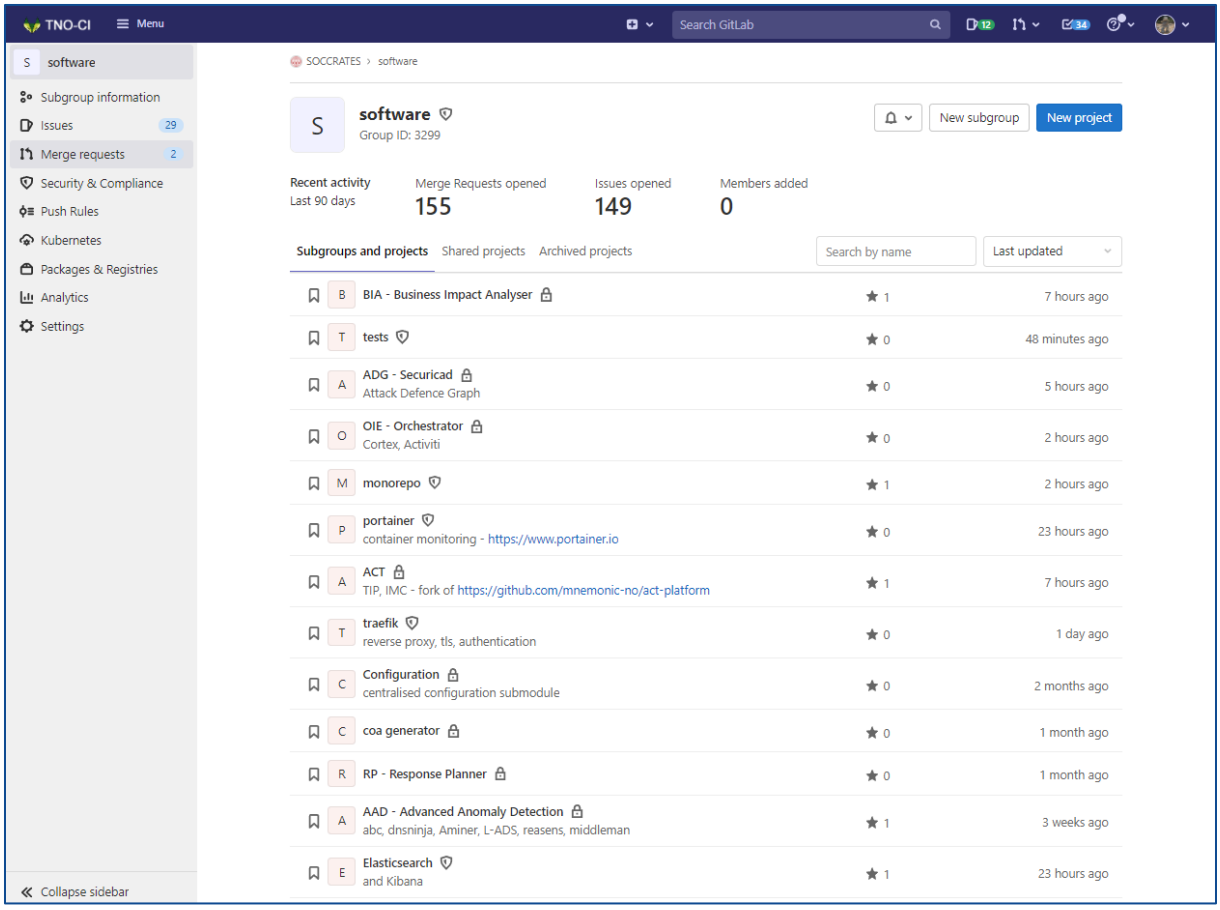


Figure 4-1 : SOCCRATES GitLab software repository

[8] https://github.com/anchore/grype

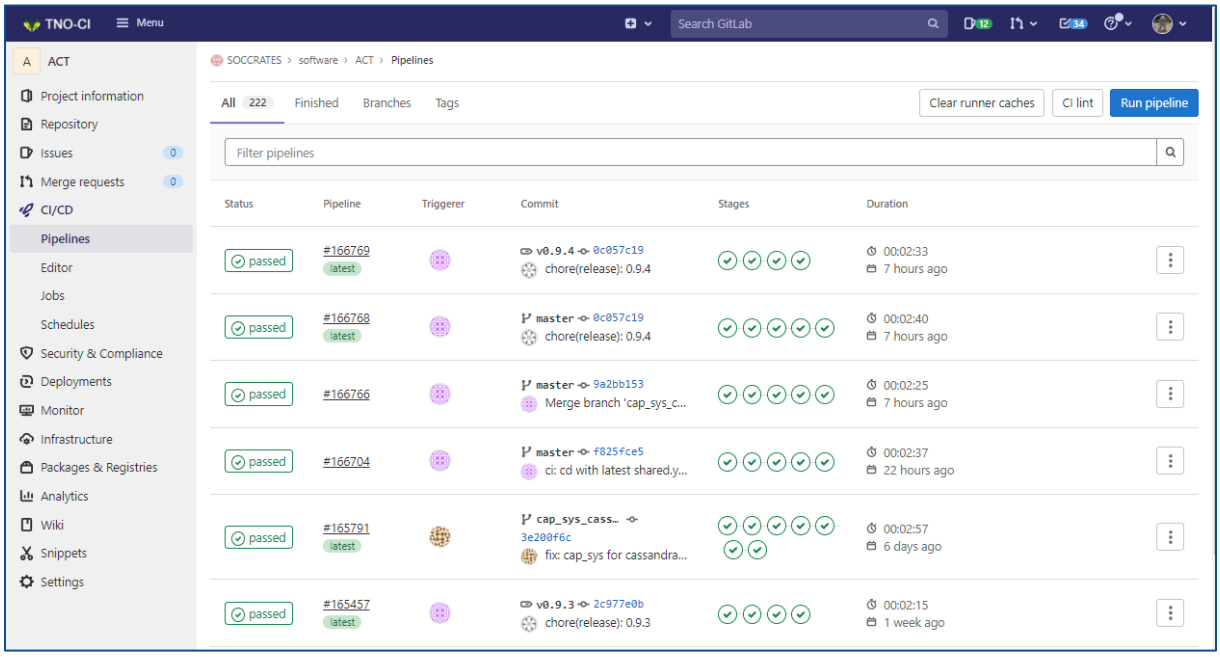


Figure 4-2 : SOCCRATES GitLab CI/CD pipelines for ACT

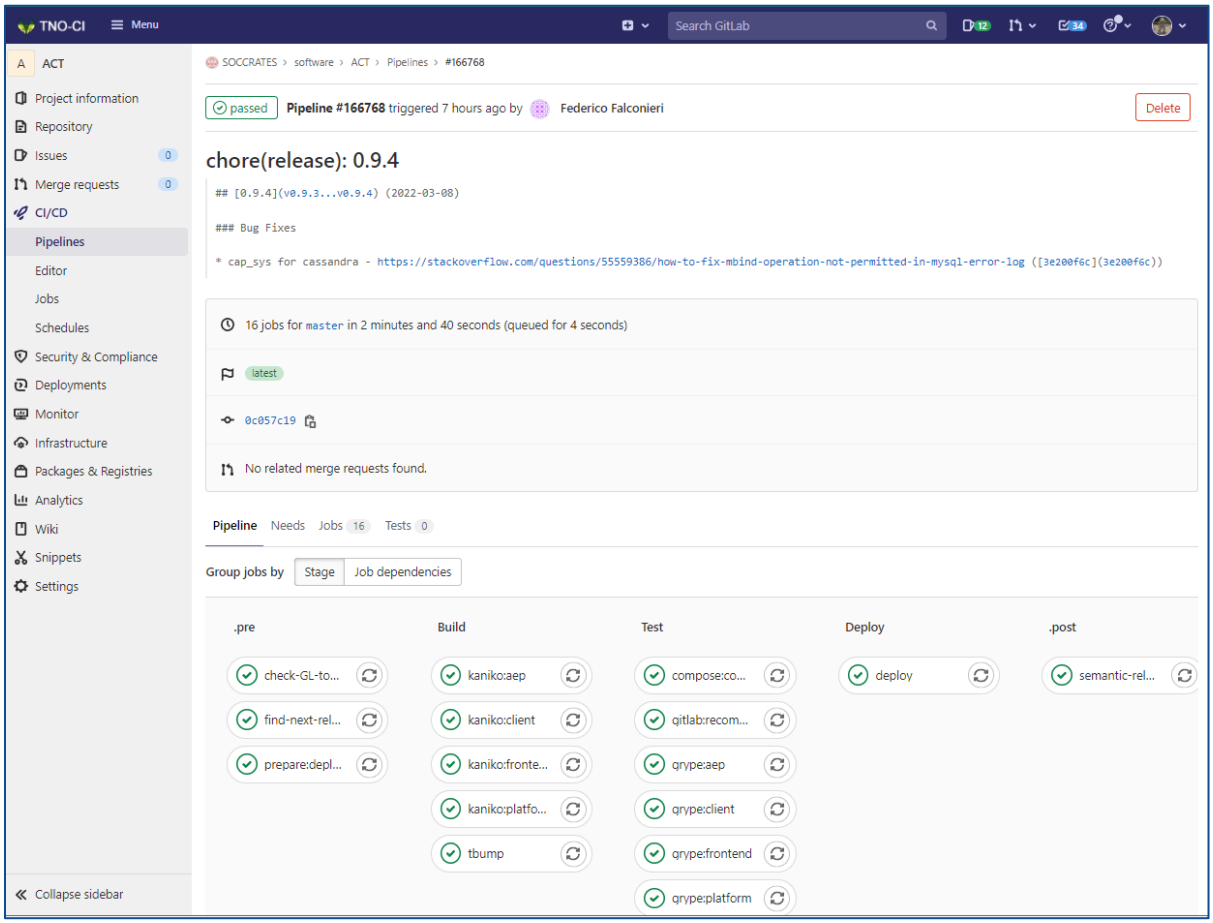


Figure 4-3 : SOCCRATES GitLab CI/CD pipeline details for ACT, including vulnerability tests with Grype

4.3 Container monitoring

A Portainer[9] deployment is used to monitor the deployed components in the staging environment. A Docker Swarm portainer deployment is included with the rest of the software for ease of use at the pilot sites. Portainer allows to check container up-time, logs, environment variables and more. This valuable information allows the developer to establish if their application not only deploys but also works, to an extent, as intended. The dashboard of Portainer is depicted in Figure 4-4. The current SOCCRATES Platform deployment consists of 55 Docker containers. Using the Swarm visualiser, it is possible to visualise the deployment on the virtualised hosts. In Figure 4-5 the deployment is visualised on four virtualised hosts.

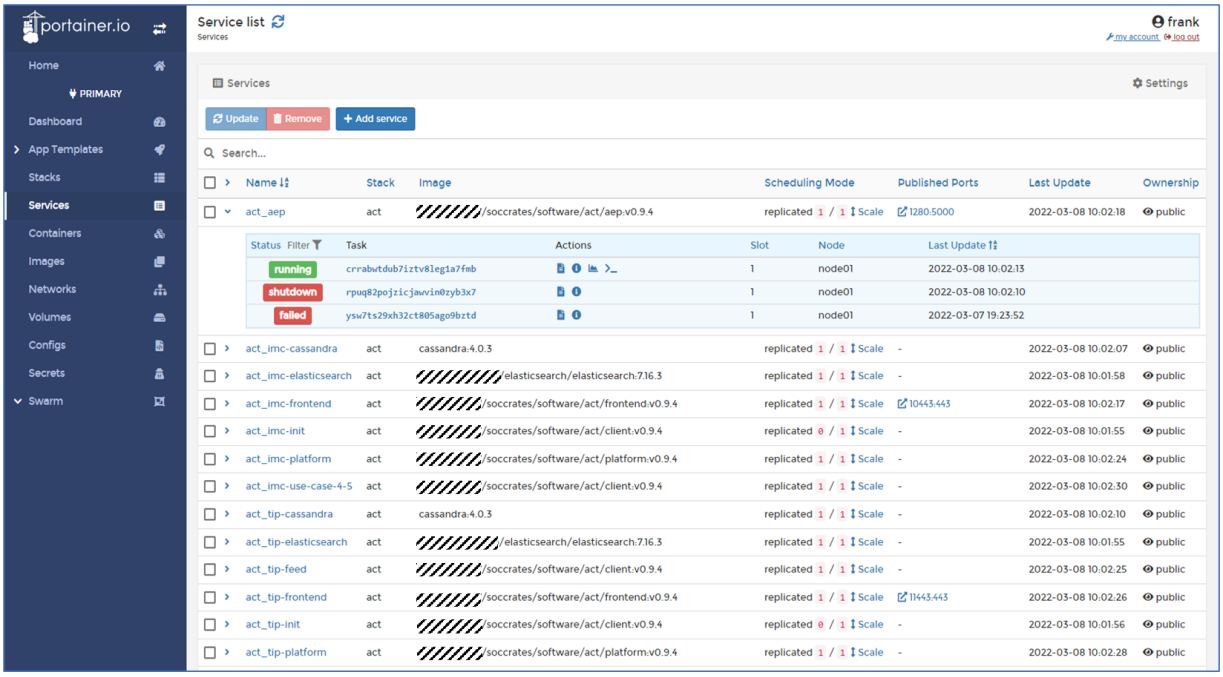


Figure 4-6 is a view of Portainer status view of the SOCCRATES Platform services.

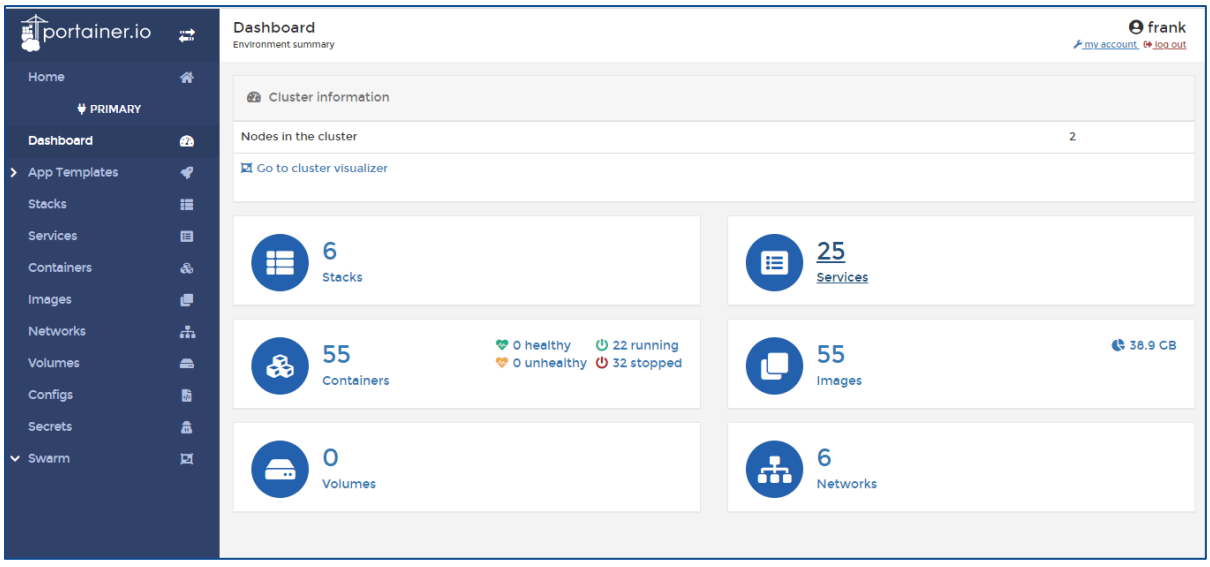


Figure 4-4 : SOCCRATES Portainer at deployment testing environment

[9] https://www.portainer.io

Classification level: Public

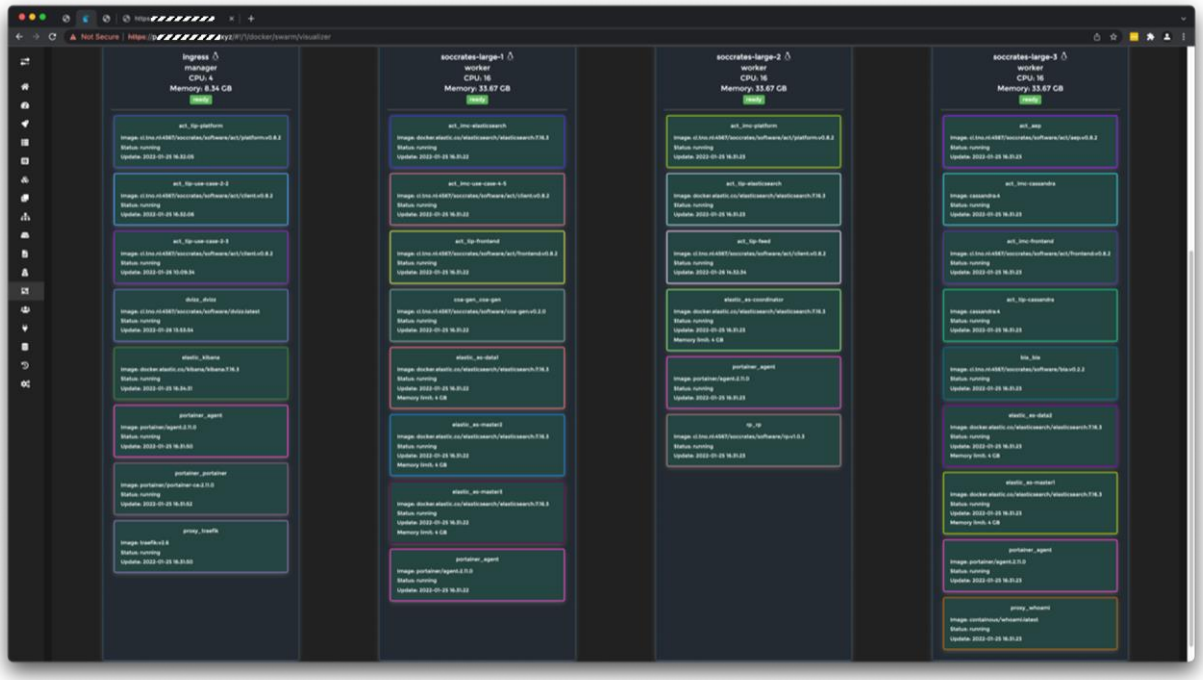


Figure 4-5 : SOCCRATES platform deployment visualised in Portainer with Swarm visualizer

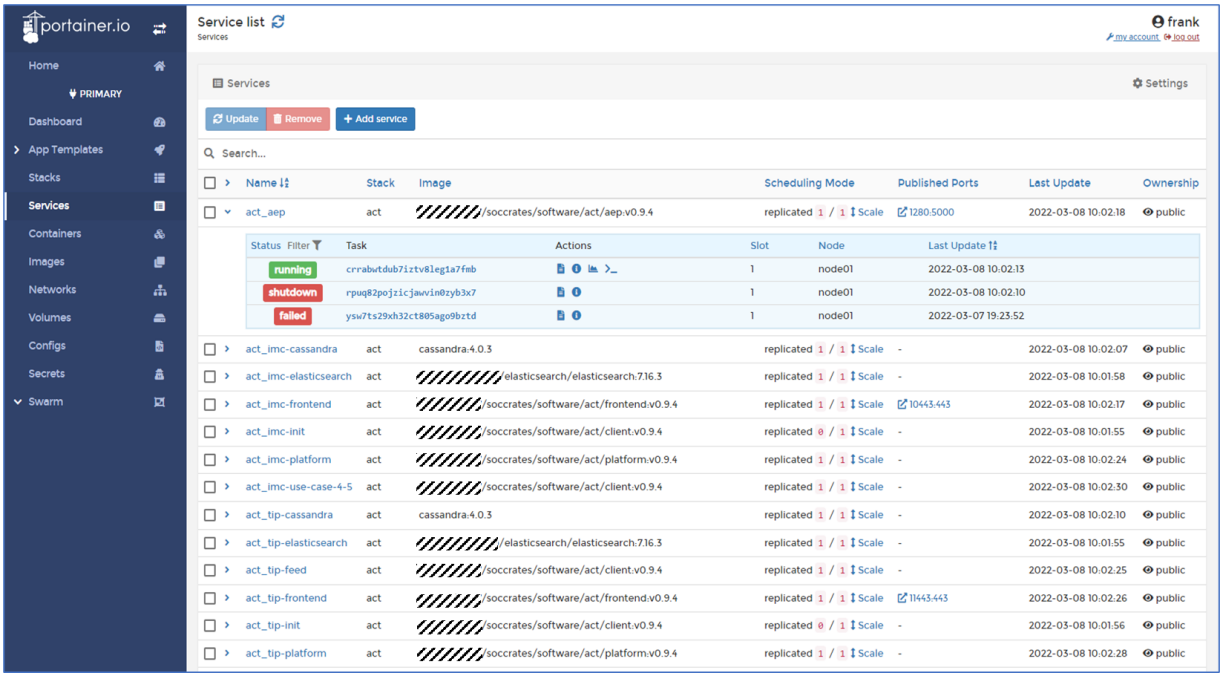


Figure 4-6 : SOCCRATES platform deployment visualised in Portainer with Service list

4.4 Log4j testing

A critical vulnerability in the Log4j logging utility was announced in December 2021. Log4j is widely used by many large open source projects, so we suspected that the SOCCRATES components, like Elasticsearch, would be affected as well. We designed a custom Gitlab CI job using grype[10] to test our container images. The job would fail if any of the Log4j vulnerabilities was detected. We added this

[10] https://github.com/anchore/grype

test to all existing Continuous Integration pipelines and then alerted the developers. In this way we were able to identify, quickly reduce the number of affected images and ensure no new Log4j vulnerabilities are introduced.

5 Overall results and conclusion

5.1 Test Execution

At the end of 2021, we identified that preparing the deliverable D6.3 before the end of January 2022 would be difficult as the deployment at the two pilot sites would start in January 2022 and all our effort would have to be put into deployment. This is why we asked our EU Project Officer to postpone the deadline of this deliverable to end of March 2022. This was good as we experienced additional delays in the execution of the integration and deployment testing compared to the test plan described in section 2.4. In Figure 5.1 the actual timelines of the execution of the integration and deployment tests performed by task 6.3 are depicted.

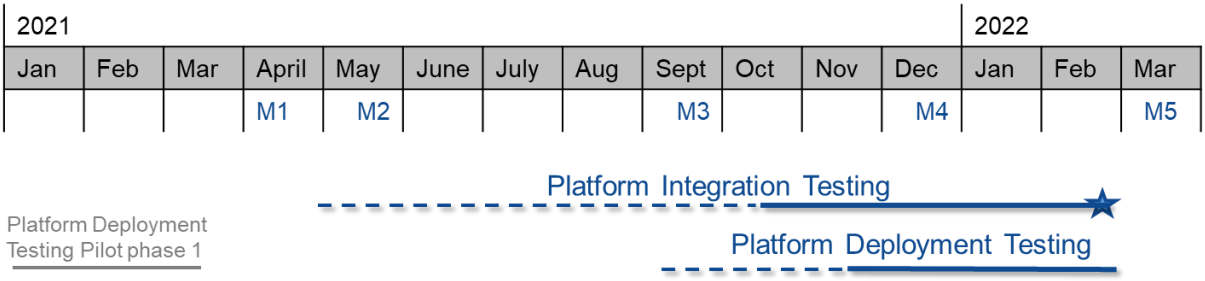


Figure 5.1 : Actual test execution

The reasons for the delays were the following:

- All partners pushed their software developments to be finished early December, such that they could be Dockerised and integrated in the CI/CD pipeline at TNO. Most software was ready in December 2021 for deployment testing. In December 2022 and first weeks of 2022 all SOCCRATES components were Dockerised and deployment tests were successfully executed. For some software components it was known that additional functionality was still being added. This could be managed by pushing the updated code in the CI/CD pipeline. Although integration testing had taken place, it was only in beginning of 2022 when the full integration tests of all SOCCRATES components for all the use cases became possible. This led to identification of some new issues that needed to be fixed (including changes to the APIs) and tested.
- Parallel to the final integration and deployment testing activities, the deployment at the two pilot sites had started. The people that were doing the software development and testing also needed to support the deployment at the pilot sites. Due to Covid travel restrictions this could not be done at pilot sites by the partners that developed the code. The fact that we had to organize the support for the deployment of the SOCCRATES platform at the two pilot sites remotely , introduced additional delays for the testing activities of task 6.3.
- During the deployment testing phase the Log4j vulnerability was reported. Several SOCCRATES components had Log4J vulnerabilities and needed to be fixed. This again resulted in some additional delays due to bug fixing. Note that most of these Log4j vulnerabilities were in the underlying (open source) software that is used within a SOCCRATES component. Additionally, Docker containers of the SOCCRATES components with the Log4j vulnerabilities were not allowed to run in the TNO deployment test environment, causing further delays for deployment testing. As described in section 4.4 we added specific Log4j tests into the CI/CD pipeline.

Overall, this resulted in a two months delay for deployment testing. As this deliverable was already delayed for 2 months, no additional extension of the due date for this deliverable was needed.

5.2 Overall test results

All the SOCCRATES integration tests have been successfully completed. This included five tests for triggering each of the SOCCRATES use cases, eleven tests for the integration of Activiti and Cortex, tests for integration Cortex Analysers / Responders with specific components, and two tests for interaction with the Shadowserver API.

For the deployment testing a deployment testing environment has been setup with a CI/CD pipeline that automatically performs tests and deploys the SOCCRATES components as Docker containers in the virtualised staging environment. The execution and provisioning of the services by the Docker containers is monitored by Portainer. All SOCCRATES components have been successfully Dockerised and executed in the staging environment. The SOCCRATES partners have been provided access to the web-based services provided by the Docker containers of the SOCCRATES components, and have successful verified correct deployment of their SOCCRATES component.

The next step of the project is to deploy the SOCCRATES platform at two pilot sites and evaluate the SOCCRATES platform in a relevant operational environment. The results of the pilot will be described D7.2 Pilot evaluation report.

6 References

[1] Frank Fransen, SOCCRATES D2.3 - System Architecture & Interface Specification - Final version, August 2021.

[2] Seamus Moloney, SOCCRATES D3.3 - Specification of the common ICT infrastructure reference meta model. August 2021.

[3] Jesús Villalobos Nieto, SOCCRATES D6.2 - Initial version of the SOCCRATES platform, November 2021.

7 Abbreviations

This glossary serves as inventory of abbreviations used in the document.

Acronym	Description
ACT	semi-Automated Cyber Threat intelligence
ADG	Attack Defence Graph
AEF	Argus Event Format
AI	Artificial Intelligence
AIT	AIT Austrian Institute of technology
API	Application Programming Interface
APT	Advance Persistent Threat
ATOS	ATOS Spain
AV	AntiVirus
BPMN	Business Process Model and Notation
CI/CD	Continuous Integration and Continuous Deployment[11]
CC	Command and Control
CERT	Computer Emergency Response Team
CMDB	Configuration Management Database
CSIRT	Computer Security Incident Response Team
CoA	Course of Action
CTI	Cyber Threat Intelligence
DC	DataCentre
DGA	Domain Generated Algorithm
DNS	Domain Name System
EDR	Endpoint Detection and Response
ELK	Elasticsearch/Logstash/Kibana
FRS	Foreseeti
FSC	F-secure
ICT	Information and Communication Technology
IDS	Intrusion Detection System
IMC	Infrastructure Modelling Component
IMT	Institut Mines Télécom – Télécom SudParis
INTF	Interface
IoC	Indicators of Compromise
IP	Internet Protocol
IPS	Intrusion Prevention System
IRM	Incident Response and Management
ITIL	Information Technology Infrastructure Library
KTH	Kungliga Tekniska högskolan - Royal Institute of Technology
LAN	Local Area Network
LDAP	Lightweight Directory Access Protocol
M_n	Infrastructure Model (at time *n*)
MNM	mnemonic
MSSP	Managed Security Service Provider

[11] It is common that the CD refers to Continuous Delivery. In this report, we prefer Continuous Deployment.

MTTD	Mean Time To Detection
NOC	Network Operations Centre
OT	Operational Technology
OS	Operating System
RORI	Return on Response Investment
SDN	Software Defined Network
SHS	Shadowserver
SIEM	Security information and event management
SOAR	Security Orchestration, Automation and Response
SOC	Security Operation Centre
SOCCRATES	SOC & CSIRT Response to Attacks & Threats based on attack defence graph Evaluation Systems
SSL	Secure Sockets Layer
TAP	Test Access Point
TIP	Threat Intelligence platform
TLS	Transport Layer Security
TNO	Nederlandse Organisatie voor toegepast natuurwetenschappelijk onderzoek
TTC	Time To Compromise
UC	Use Case
VLAN	Virtual LAN
VM	Virtual Machine
VTF	Vattenfall

Appendix A. Integration Test Cases

Test Case ID	A.1	**Components**	
Type of testing	Integration	**Web Frontend**	Orchestrator
Description	Triggering UC1 (SOCCRATES Orchestrator Web Frontend)		
Specification		**Priority**	Low/Medium/High
Prepared by	Susana González	**Tested by**	Susana González
Pre-condition(s)			
Test steps			
1	Open SOCCRATES Web Frontend and manually trigger a new SOCCRATES UC. Select UC type « UC1- New Incident »		
2			
3			
Input data	{"SourceIP": "10.255.129.220", "SourcePort": "1234", "DestinationIP": "10.255.131.49", "DestinationPort": "80", "Severity": "High", "Tactic": "TA0010", "Date": "2022-02-25T12:38:05.098Z"}		
Expected result	Orchestrator initiates UC1 with the IP address of the affected host as a parameter		
Test Case Result	Success		

Test Case ID	A.2	**Components**	
Type of testing	Integration	**TIP**	Orchestrator
Description	Triggering UC2 - CTI on observed exploit for vulnerability		
Specification		**Priority**	Low/Medium/High
Prepared by	Martin Eian	**Tested by**	Martin Eian
Pre-condition(s)			
Test steps			
1	In the TIP GUI, create an **exploits** fact that connects a **content** object to a **vulnerability** object		

Classification level: Public

2	
3	
Input data	CVE from vulnerability object
Expected result	Orchestrator initiates UC2 with the supplied CVE as a parameter
Test Case Result	Success

Test Case ID	A.3	**Components**	
Type of testing	Integration	**TIP**	Orchestrator
Description	Triggering UC2 - new technique associated with a threat actor		
Specification		**Priority**	Low/Medium/High
Prepared by	Martin Eian	**Tested by**	Jesús Villalobos
Pre-condition(s)			
Test steps			
1	In the TIP GUI, create an **attributedTo** fact that connects an **incident** object to a **threatActor** object		
2	In the TIP GUI, create an **observedIn** fact that connects a **technique** object to the **incident** object used in step 1		
3			
Input data	MITRE ATT&CK technique ID, threat actor name		
Expected result	Orchestrator initiates UC2 with the supplied ATT&CK technique ID and threat actor name as parameters		
Test Case Result	Success		

Test Case ID	A.4	**Components**	
Type of testing	Integration	**IMC**	Orchestrator
Description	Triggering UC3		
Specification	D3.3 Figure 3 [2]	**Priority**	Low/Medium/High

Prepared by	Martin Eian	**Tested by**	Jesús Villalobos
Pre-condition(s)			
Test steps			
1	In the IMC GUI, create an **affects** fact that connects a **vulnerability** object to a **software** object		
2			
3			
Input data	CVE from vulnerability object		
Expected result	Orchestrator initiates UC3 with the supplied CVE as a parameter		
Test Case Result	Success		

Test Case ID	A.5	**Components**	
Type of testing	Integration	**IMC**	Orchestrator
Description	Triggering UC4		
Specification	D3.3 Figure 3 [2]	**Priority**	Low/Medium/High
Prepared by	Martin Eian	**Tested by**	Jesús Villalobos
Pre-condition(s)			
Test steps			
1	In the IMC GUI, create a **configuredOn** fact that connects a **ruleSet** object to a **host** object		
2	In the IMC GUI, create a **componentOf** fact that connects a **rule** object to a **ruleSet** object		
3			
Input data	host		
Expected result	Orchestrator initiates UC4 with the host where the configuration change took place as a parameter		
Test Case Result	Success		

Test Case ID	A.6	**Components**

Type of testing	Integration	**IMC**	Orchestrator
Description	Triggering UC5		
Specification	D3.3 Figure 3 [2]	**Priority**	Low/Medium/High
Prepared by	Martin Eian	**Tested by**	Jesús Villalobos
Pre-condition(s)			
Test steps			
1	In the IMC GUI, create an **assignedTo** fact that connects a new (non-existing) **ipv4** object to a new (non-existing) **host** object.		
2			
3			
Input data	host		
Expected result	Orchestrator initiates UC5 with the supplied host as a parameter		
Test Case Result	Success		

Test Case ID	B.1	**Components**	
Type of testing	Integration	**Orchestrator**	SOCCRATES_ADG_ANALYZER_1_0
Description	Test the invocation of the Cortex Analyser from the Orchestrator Core module		
Specification	Deliverable D6.2 section 2.2.1 [3]	**Priority**	High
Prepared by	Jesús Villalobos	**Tested by**	Jesús Villalobos
Pre-condition(s)	Cortex and ADG services are up and running so the test can invoke the Analyser		
Test steps			
1	Execute the JUnit test		
2	Orchestrator Core automatically configures and performs the invocation of the Analyser		
3	Cortex receives the request from the Orchestrator Core and communicates with the ADG service		
4	The ADG Analyser responds to the request and pass the result to the Orchestrator		

5	Orchestrator takes the JSON response and tries to map it into the given data model class to validate the response and finish the test

Input data	{
	"date": "2022-02-22",
	"threatActor": {
	"threatActor": "APT-28",
	"capabilities": ["T1110.002","T1087"]
	},
	"entryPoints": [{"ref": "3a33fbe6-8c94-4be6-aca6-e3f13c3f64e3","type": "TA0010"}],
	"highValueAssets": [
	{
	"type": "Confidentiality",
	"ref": "3a33fbe6-8c94-4be6-aca6-e3f13c3f64e3"
	},
	{
	"type": "Integrity",
	"ref": "3a33fbe6-8c94-4be6-aca6-e3f13c3f64e3"
	},
	{
	"type": "Availability",
	"ref": "3a33fbe6-8c94-4be6-aca6-e3f13c3f64e3"
	}
	]
	}

Expected result	The Orchestrator successfully maps the request JSON into a Java Object. The Analyser returns valid JSON data in the expected format (as described in deliverable D2.3) and the Orchestrator correctly maps it into a Java object of the expected class
Test Case Result	Success

Test Case ID	B.2	**Components**	
Type of testing	Integration	**Orchestrator**	SOCCRATES_Coa_Generator_1_0
Description	Test the invocation of the Cortex Analyser from the Orchestrator Core module		
Specification	Deliverable D6.2 section 2.2.1 [3]	**Priority**	High
Prepared by	Jesús Villalobos	**Tested by**	Jesús Villalobos

Pre-condition(s)	Cortex and CoA services are up and running so the test can invoke the Analyser
Test steps	
1	Execute the JUnit test
2	Orchestrator Core automatically configures and performs the invocation of the Analyser
3	Cortex receives the request from the Orchestrator Core and communicates with the CoA service
4	The CoA Analyser responds to the request and pass the result to the Orchestrator
5	Orchestrator takes the JSON response and tries to map it into the given data model class to validate the response and finish the test

Input data	{
	"simulationId": "307666084811239"
	}

Expected result	The Orchestrator successfully maps the request JSON into a Java Object. The Analyser returns valid JSON data in the expected format (as described in deliverable D2.3) and the Orchestrator correctly maps it into a Java object of the expected class
Test Case Result	Success

Test Case ID	B.3	**Components**	
Type of testing	Integration	**Orchestrator**	SOCCRATES_IMC_ANALYZER_1_0
Description	Test the invocation of the Cortex Analyser from the Orchestrator Core module		
Specification	Deliverable D6.2 section 2.2.1 [3]	**Priority**	High
Prepared by	Jesús Villalobos	**Tested by**	Jesús Villalobos
Pre-condition(s)	Cortex and IMC services are up and running so the test can invoke the Analyser		
Test steps			
1	Execute the JUnit test		
2	Orchestrator Core automatically configures and performs the invocation of the Analyser		
3	Cortex receives the request from the Orchestrator Core and communicates with the IMC service		

4	The IMC Analyser responds to the request and pass the result to the Orchestrator
5	Orchestrator takes the JSON response and tries to map it into the given data model class to validate the response and finish the test

Input data	{
	"ipv4": "10.255.131.163"
	}

Expected result	The Orchestrator successfully maps the request JSON into a Java Object. The Analyser returns valid JSON data in the expected format (as described in deliverable D2.3) and the Orchestrator correctly maps it into a Java object of the expected class
Test Case Result	Success

Test Case ID	B.4	**Components**	
Type of testing	Integration	**Orchestrator**	Mailer_1_0
Description	Test the invocation of the Cortex Analyser from the Orchestrator Core module		
Specification	Deliverable D6.2 section 2.2.1 [3]	**Priority**	High
Prepared by	Jesús Villalobos	**Tested by**	Jesús Villalobos
Pre-condition(s)	Cortex service is up and running so the test can invoke the Analyser		
Test steps			
1	Execute the JUnit test		
2	Orchestrator Core automatically configures and performs the invocation of the Analyser		
3	Cortex receives the request from the Orchestrator Core and sends the email		
4	Cortex responds to the request with a successfull message		

Input data	{
	"description": "Dear Analyst, ...",
	"title": "CoAs suggested by SOCCRATES platform as Response on Detected Ongoing Attack (UC-1)",
	"tags": ["mail:soccrates.manager@gmail.com"],
	"username": "soccrates.manager@gmail.com"
	}		

Expected result	The Orchestrator successfully maps the request JSON into a Java Object. Cortex responds to the request with a 200 OK message and the Cortex job ends succesfully. Also the email arrives correctly to the mailbox.
Test Case Result	Success

Test Case ID	B.5	**Components**	
Type of testing	Integration	**Orchestrator**	SOCCRATES_Business_Get-Critical_Analyzer_1_0
Description	Test the invocation of the Cortex Analyser from the Orchestrator Core module		
Specification	Deliverable D6.2 section 2.2.1 [3]	**Priority**	High
Prepared by	Jesús Villalobos	**Tested by**	Jesús Villalobos
Pre-condition(s)	Cortex and BIA services are up and running so the test can invoke the Analyser		
Test steps			
1	Execute the JUnit test		
2	Orchestrator Core automatically configures and performs the invocation of the Analyser		
3	Cortex receives the request from the Orchestrator Core and communicates with the BIA service		
4	The BIA Analyser responds to the request and pass the result to the Orchestrator		
5	Orchestrator takes the JSON response and tries to map it into the given data model class to validate the response and finish the test		
Input data	{ "Threshold": 0.1 }		
Expected result	The Orchestrator successfully maps the request JSON into a Java Object. The Analyser returns valid JSON data in the expected format (as described in deliverable D2.3) and the Orchestrator correctly maps it into a Java object of the expected class		
Test Case Result	Success		

Test Case ID	B.6	**Components**

Type of testing	Integration	**Orchestrator**	SOCCRATES_Business_Impact_analyzer_1_0
Description	Test the invocation of the Cortex Analyser from the Orchestrator Core module		
Specification	Deliverable D6.2 section 2.2.1 [3]	**Priority**	High
Prepared by	Jesús Villalobos	**Tested by**	Jesús Villalobos
Pre-condition(s)	Cortex and BIA services are up and running so the test can invoke the Analyser		
Test steps			
1	Execute the JUnit test		
2	Orchestrator Core automatically configures and performs the invocation of the Analyser		
3	Cortex receives the request from the Orchestrator Core and communicates with the BIA service		
4	The BIA Analyser responds to the request and pass the result to the Orchestrator		
5	Orchestrator takes the JSON response and tries to map it into the given data model class to validate the response and finish the test		

Input data	{
	"attack_time": "2021-12-02T11:38:05.098Z",
	"Impacted_assets": [
	{
	"assetID": "3a33fbe6-8c94-4be6-aca6-e3f13c3f64e3",
	"confidentiality": {
	"ttc": 0,
	"probability": 1.0
	},
	"integrity": {
	"ttc": 0,
	"probability": 1.0
	},
	"availability": {
	"ttc": 0,
	"probability": 1.0
	}
	}]}		

Expected result	The Orchestrator successfully maps the request JSON into a Java Object. The Analyser returns valid JSON data in the expected format (as described in deliverable D2.3) and the Orchestrator correctly maps it into a Java object of the expected class
Test Case Result	Success

Test Case ID	B.7	**Components**	
Type of testing	Integration	**Orchestrator**	SOCCRATES_Business_Impact_responder_1_0
Description	Test the invocation of the Cortex Responder from the Orchestrator Core module		
Specification	Deliverable D6.2 section 2.2.1 [3]	**Priority**	High
Prepared by	Jesús Villalobos	**Tested by**	Jesús Villalobos
Pre-condition(s)	Cortex and BIA services are up and running so the test can invoke the Analyser		
Test steps			
1	`Execute the JUnit test`		
2	`Orchestrator Core automatically configures and performs the invocation of the Responder`		
3	`Cortex receives the request from the Orchestrator Core and communicates with the BIA service`		
4	`The BIA Analyser responds to the request and returns a 200 OK`		
Input data	{ "newAsset": ["atos-host-1"] }		
Expected result	The Orchestrator successfully maps the request JSON into a Java Object. The Orchestrator successfully maps the request JSON into a Java Object. The Responder returns a 200 OK and the Cortex job ends succesfully		
Test Case Result	Success		

Test Case ID	B.8	**Components**	
Type of testing	Integration	**Orchestrator**	SOCCRATES_RP_COA_RANKING_ANALYZER_1_2
Description	Test the invocation of the Cortex Analyser from the Orchestrator Core module		

Specification	Deliverable D6.2 section 2.2.1 [3]	**Priority**	High
Prepared by	Jesús Villalobos	**Tested by**	Jesús Villalobos
Pre-condition(s)	Cortex and RP services are up and running so the test can invoke the Analyser		
Test steps			
1	Execute the JUnit test		
2	Orchestrator Core automatically configures and performs the invocation of the Analyser		
3	Cortex receives the request from the Orchestrator Core and communicates with the RP service		
4	The RP Analyser responds to the request and pass the result to the Orchestrator		
5	Orchestrator takes the JSON response and tries to map it into the given data model class to validate the response and finish the test		
Input data			
Expected result	The Orchestrator successfully maps the request JSON into a Java Object. The Analyser returns valid JSON data in the expected format (as described in deliverable D2.3) and the Orchestrator correctly maps it into a Java object of the expected class		
Test Case Result	Success		

Test Case ID	B.9	**Components**	
Type of testing	Integration	**Orchestrator**	SOCCRATES_RP_CONTAINMENT_COA_GENERATOR_1_0
Description	Test the invocation of the Cortex Analyser from the Orchestrator Core module		
Specification	Deliverable D6.2 section 2.2.1 [3]	**Priority**	High
Prepared by	Jesús Villalobos	**Tested by**	Jesús Villalobos
Pre-condition(s)	Cortex and RP services are up and running so the test can invoke the Analyser		
Test steps			
1	Execute the JUnit test		
2	Orchestrator Core automatically configures and performs the invocation of the Analyser		

3	Cortex receives the request from the Orchestrator Core and communicates with the RP service
4	The RP Analyser responds to the request and pass the result to the Orchestrator
5	Orchestrator takes the JSON response and tries to map it into the given data model class to validate the response and finish the test
Input data	{ "siem_label": "TA0010", "sourceIPs": ["10.255.131.163"], "destinationIPs": ["10.255.130.79"], "type": "containment" }
Expected result	The Orchestrator successfully maps the request JSON into a Java Object. The Analyser returns valid JSON data in the expected format (as described in deliverable D2.3) and the Orchestrator correctly maps it into a Java object of the expected class
Test Case Result	Success

Test Case ID	B.10	**Components**	
Type of testing	Integration	**Orchestrator**	SOCCRATES_RP_CONTAINMENT_COA_GENERATOR_1_0
Description	Test the invocation of the Cortex Analyser from the Orchestrator Core module		
Specification	Deliverable D6.2 section 2.2.1 [3]	**Priority**	High
Prepared by	Jesús Villalobos	**Tested by**	Jesús Villalobos
Pre-condition(s)	Cortex and RP services are up and running so the test can invoke the Analyser		
Test steps			
1	Execute the JUnit test		
2	Orchestrator Core automatically configures and performs the invocation of the Analyser		
3	Cortex receives the request from the Orchestrator Core and communicates with the RP service		
4	The RP Analyser responds to the request and pass the result to the Orchestrator		

5	Orchestrator takes the JSON response and tries to map it into the given data model class to validate the response and finish the test

Input data	{
	"siem_label": "TA0010",
	"sourceIPs": ["10.255.131.163"],
	"destinationIPs": ["10.255.130.79"],
	"type": "stop_exfiltration"
	}

Expected result	The Orchestrator successfully maps the request JSON into a Java Object. The Analyser returns valid JSON data in the expected format (as described in deliverable D2.3) and the Orchestrator correctly maps it into a Java object of the expected class
Test Case Result	Success

Test Case ID	B.11	**Components**	
Type of testing	Integration	**Orchestrator**	SOCCRATES_AAD_RESPONDER_1_0
Description	Test the invocation of the Cortex Responder from the Orchestrator Core module		
Specification	Deliverable D6.2 section 2.2.1 [3]	**Priority**	High
Prepared by	Jesús Villalobos	**Tested by**	Jesús Villalobos
Pre-condition(s)	Cortex and AAD services are up and running so the test can invoke the Responder		
Test steps			
1	Execute the JUnit test		
2	Orchestrator Core automatically configures and performs the invocation of the Responder		
3	Cortex receives the request from the Orchestrator Core and communicates with the AAD service		
4	The AAD Responder responds to the request with a 200 OK code		
Input data	{ "timestamp": "2020-12-12T16:45:14Z", "type": "newVulnerability", "host": "computer01.soccrates.eu",		

	"vulnerabilities": [{"software": "cpe:2.3:a:google:chrome:*:*:*:*:*:*:*:*","cve": "CVE-2021-21112"}] }
Expected result	The Orchestrator successfully maps the request JSON into a Java Object. The Responder returns a 200 OK and the Cortex job ends succesfully
Test Case Result	Success

Test Case ID	C.1	**Components**	
Type of testing	Integration	**Orchestrator**	SOCCRATES_IMC_ANALYZER_1_0
Description	IMC Cortex Analyzer integration with IMC		
Specification	D2.3 Figure 4.3 [1]	**Priority**	Low/Medium/High
Prepared by	Susana González	**Tested by**	Susana González
Pre-condition(s)	A host with the ip address included in the test has been registered in the IMC		
Test steps			
1	`Cortex API to run an analyzer is invoked from pytest script with the id of the IMC Analizer and the request data.`		
2	`The job id of the execution is retrieved and it is waited a timeout for a response.`		
3	`Cortex API to get the result of a job is invoked and the response is analyzed to check if there is a report and success is True.`		
Input data	`{"ipv4":"10.255.131.163"}`		

Expected result	IMC Analyzer returns the UUID associated to the asset with the ipv4 provided
	{
	"summary": {},
	"full": {
	"host": "ca5a9a54-f3f5-4dfe-aa78-9d1eb0ab2f2a"
	},
	"success": true,
	"artifacts": [],
	"operations": []
	}		

Test Case Result	Success		

Test Case ID	C.2	Components	
Type of testing	Integration	Orchestrator	SOCCRATES_ADG_ANALYZER_1_0
Description	ADG Cortex Analyzer integration with ADG Analyzer with entry points		
Specification	D2.3 Figure 4.11, Figure 4.4 [1]	Priority	Low/Medium/High
Prepared by	Susana González	Tested by	Susana González
Pre-condition(s)	• A host with the reference included in the request data has been registered in the IMC • The analyzer has been configured correctly with the url, username and password to access the TIP and IMC		
Test steps			
1	Cortex API to run an analyzer is invoked from pytest script with the id of the ADG Analizer and the request data.		
2	The job id of the execution is retrieved and it is waited a timeout for a response.		
3	Cortex API to get the result of a job is invoked and the response is analyzed to check if there is a report and success is True.		
Input data	{"date":"2022-02-22","entryPoints":[{"ref":"3a33fbe6-8c94-4be6-aca6-e3f13c3f64e3","type":"TA0010"}],"highValueAssets":[{"type":"Confidentiality","ref":"3a33fbe6-8c94-4be6-aca6-e3f13c3f64e3"},{"type":"Integrity","ref":"3a33fbe6-8c94-4be6-aca6-e3f13c3f64e3"},{"type":"Availability","ref":"3a33fbe6-8c94-4be6-aca6-e3f13c3f64e3"}]}		
Expected result	ADG analysis (screenshot of the result has not been included due to its extension)		
Test Case Result	Success		

Test Case ID	C.3	Components	
Type of testing	Integration	Orchestrator	SOCCRATES_ADG_ANALYZER_1_0
Description	ADG Cortex Analyzer integration with ADG Analyzer with assets of interest		
Specification	D2.3 Figure 4.11, Figure 4.4 [1]	Priority	Low/Medium/High
Prepared by	Susana González	Tested by	Susana González

Pre-condition(s)	• A host with the reference included in the request data has been registered in the IMC • The analyzer has been configured correctly with the url, username and password to access the TIP and IMC
Test steps	
1	`Cortex API to run an analyzer is invoked from pytest script with the id of the ADG Analizer and the request data.`
2	`The job id of the execution is retrieved and it is waited a timeout for a response.`
3	`Cortex API to get the result of a job is invoked and the response is analyzed to check if there is a report and success is True.`
Input data	`{"date":"2022-02-22","assetsOfInterest":[{"ref":"3a33fbe6-8c94-4be6-aca6-e3f13c3f64e3”}],"highValueAssets":[{"type":"Confidentiality","ref":"3a33fbe6-8c94-4be6-aca6-e3f13c3f64e3"},{"type":"Integrity","ref":"3a33fbe6-8c94-4be6-aca6-e3f13c3f64e3"},{"type":"Availability","ref":"3a33fbe6-8c94-4be6-aca6-e3f13c3f64e3"}]}`
Expected result	ADG analysis (screenshot of the result has not been included due to its extension)
Test Case Result	Success

Test Case ID	C.4	**Components**	
Type of testing	Integration	**Orchestrator**	SOCCRATES_ADG_ANALYZER_1_0
Description	ADG Cortex Analyzer integration with ADG Analyzer with vulnerabilities		
Specification	D2.3 Figure 4.11, Figure 4.4 [1]	**Priority**	Low/Medium/High
Prepared by	Susana González	**Tested by**	Susana González
Pre-condition(s)	• A host with the reference included in the request data has been registered in the IMC • The analyzer has been configured correctly with the url, username and password to access the TIP and IMC		
Test steps			
1	`Cortex API to run an analyzer is invoked from pytest script with the id of the ADG Analizer and the request data.`		
2	`The job id of the execution is retrieved and it is waited a timeout for a response.`		

3	Cortex API to get the result of a job is invoked and the response is analyzed to check if there is a report and success is True.
Input data	{"date":"2022-02-22","vulnerabilities":[{"ref":"3a33fbe6-8c94-4be6-aca6-e3f13c3f64e3","cve":"CVE-2021-1234"}],"high-ValueAssets":[{"type":"Confidentiality","ref":"3a33fbe6-8c94-4be6-aca6-e3f13c3f64e3"},{"type":"Integrity","ref":"3a33fbe6-8c94-4be6-aca6-e3f13c3f64e3"},{"type":"Availabil-ity","ref":"3a33fbe6-8c94-4be6-aca6-e3f13c3f64e3"}]}
Expected result	ADG analysis (screenshot of the result has not been included due to its extension)
Test Case Result	Success

Test Case ID	C.5	**Components**	
Type of testing	Integration	**Orchestrator**	SOCCRATES_ADG_ANALYZER_1_0
Description	ADG Cortex Analyzer integration with ADG Analyzer with threat actor		
Specification	D2.3 Figure 4.11, Figure 4.4 [1]	**Priority**	Low/Medium/High
Prepared by	Susana González	**Tested by**	Susana González
Pre-condition(s)	• A host with the reference included in the request data has been registered in the IMC • The analyzer has been configured correctly with the url, username and password to access the TIP and IMC		
Test steps			
1	Cortex API to run an analyzer is invoked from pytest script with the id of the ADG Analizer and the request data.		
2	The job id of the execution is retrieved and it is waited a timeout for a response.		
3	Cortex API to get the result of a job is invoked and the response is analyzed to check if there is a report and success is True.		
Input data	{"date":"2022-02-22","threatActor":{"threatActor":"APT-28","capa-bilities":["T1110.002","T1087"]},"entryPoints":[{"ref":"3a33fbe6-8c94-4be6-aca6-e3f13c3f64e3","type":"TA0010"}],"high-ValueAssets":[{"type":"Confidentiality","ref":"3a33fbe6-8c94-4be6-aca6-e3f13c3f64e3"},{"type":"Integrity","ref":"3a33fbe6-8c94-4be6-aca6-e3f13c3f64e3"},{"type":"Availabil-ity","ref":"3a33fbe6-8c94-4be6-aca6-e3f13c3f64e3"}]}		
Expected result	ADG analysis (screenshot of the result has not been included due to its extension)		
	Success		

Test Case Result	

Test Case ID	C.6	Components	
Type of testing	Integration	Orchestrator	SOCCRATES_Business_Impact_analyzer_1_0
Description	BIA Cortex Analyzer integration with Business Impact service to get impact probability		
Specification	D2.3 Figure 4.4, Figure 4.11 [1]	Priority	Low/Medium/High
Prepared by	Susana González	Tested by	Susana González
Pre-condition(s)	The asset reference included in the request data must exist in the model created from the IMC data.		
Test steps			
1	Cortex API to run an analyzer is invoked from pytest script with the id of the SOCCRATES_Business_Impact_Analyzer and the request data.		
2	The job id of the execution is retrieved and it is waited a timeout for a response.		
3	Cortex API to get the result of a job is invoked and the response is analyzed to check if there is a report and success is True.		
Input data	{"attack_time":"2021-12-02T11:38:05.098Z","Impacted_assets":[{"assetID":"3a33fbe6-8c94-4be6-aca6-e3f13c3f64e3","confidentiality":{"ttc":0, "probability":1.0},"integrity":{"ttc":0, "probability":1.0},"availability":{"ttc":0, "probability":1.0}}]}"		

Expected result	Impact analysis
	{
	"summary": {},
	"full": {
	"business_impacts": [
	{
	"business_name": "BC",
	"impact_probability": 0.46,
	"impact_criticality": 10,
	"critical_time": 0.4
	}
	],
	"report_url": "http://10.8.0.4:8003/?request=1646041831104939531"
	},
	"success": true,
	"artifacts": [],
	"operations": []
	}		

Test Case Result	Success		

Classification level: Public

Test Case ID	C.7	Components	
Type of testing	Integration	Orchestrator	SOCCRATES_Business_GetCritical_Analyzer_1_0
Description	BIA Cortex Analyzer integration with Business Impact service to get the critical assets		
Specification	D2.3 Figure 4.4, Figure 4.11 [1]	Priority	Low/Medium/High
Prepared by	Susana González	Tested by	Susana González
Pre-condition(s)	There are assets over the threshold indicated in the request data.		

Test steps	
1	Cortex API to run an analyzer is invoked from pytest script with the id of the SOCCRATES_Business_GetCritical_Analyzer and the request data.
2	The job id of the execution is retrieved and it is waited a timeout for a response.
3	Cortex API to get the result of a job is invoked and the response is analyzed to check if there is a report and success is True.

Input data	{"Threshold": 0.1}

Expected result	List of critical assets UUID references is returned :
	{
	"summary": {},
	"full": {
	"critical_assets": [
	"1ebddf2b-e47f-442c-b6ce-08f1e37ad498",
	"337c2a52-07b5-4887-80af-802233aae52d",
	"1d97d1e4-60e8-4a28-8a1f-64514a351b3b",
	"87087dbe-4ec8-4bbc-8c74-1878c65cde4b",
	"a1ea0214-386e-4308-b3ec-6cb00010e58e",
	"0ded8055-a55d-4f5e-afc6-d1a0570ebe21",
	"6111f303-0f25-4195-b31b-25d30e0b0c00",
	"383ab4e6-7645-41df-94c7-1f7e3d97df57",
	"03abd9b7-6292-4d9f-aad2-98b4dccab31c",
	"046ce13b-26fa-42c9-8f30-6e25d545713f",
	"0757a3fa-91cd-4afe-bf15-c4c37987b93b",
	"0a274fdb-c3ae-40bb-938f-dc94468b34de",
	"3a33fbe6-8c94-4be6-aca6-e3f13c3f64e3",
	"60a44bee-468f-4300-934b-eaebb4348568",
	"f1693f95-16d1-4cdc-b193-f23e283c9b50"
	],
	"report_url": "http://10.8.0.4:8003/?request=1646041802836048491"
	},
	"success": true,
	"artifacts": [],
	"operations": []
	}

Test Case Result	Success

Test Case ID	C.8	Components	
Type of testing	Integration	Orchestrator	SOC-CRATES_RP_COA_RANK-ING_ANALYZER_1_2 (Response Planner)
Description	CoA Ranking Cortex Analyzer integration with Response Planner (no contain-ment)		
Specification	D2.3 Figure 4.7, Figure 4.9 [1]	Priority	Low/Medium/High
Prepared by	Susana González	Tested by	
Pre-condition(s)	Some CoAs have been provided in the request data		
Test steps			
1	Cortex API to run an analyzer is invoked from pytest script with the id of the SOCCRATES_RP_COA_RANKING Analyzer and the request data.		
2	The job id of the execution is retrieved and it is waited a timeout for a response.		
3	Cortex API to get the result of a job is invoked and the response is analyzed to check if there is a report and success is True.		
Input data	{"si-em_label":"TA0010","containment":false,"coas":[{"defenses":[{"defense-Name":"Host isolation","defenseInfo":"Isolate the host from the network by using Network Segmenta-tion","mi-treRef":"M1030","hosts":["10.255.130.79","10.255.131.163"]},{"defense-Name":"Trafic filtering","defenseInfo":"Use network appliances to filter ingress or egress traffic and perform protocol-based filter-ing","mi-treRef":"M1037","flows":[{"src":"10.255.131.163","dst":"10.255.130.79"}]}]}]}		
Expected result			
Test Case Result	Success		

Test Case ID	C.9	Components	
Type of testing	Integration	Orchestrator	SOCCRATES_RP_COA_RANKING_ANALYZER_1_2 (Response Planner)
Description	CoA Ranking Cortex Analyzer integration with Response Planner (containment)		
Specification	D2.3 Figure 4.7, Figure 4.9 [1]	Priority	Low/Medium/High
Prepared by	Susana González	Tested by	
Pre-condition(s)	Some CoAs have been provided in the request data		
Test steps			
1	Cortex API to run an analyzer is invoked from pytest script with the id of the SOCCRATES_RP_COA_RANKING Analyzer and the request data.		
2	The job id of the execution is retrieved and it is waited a timeout for a response.		
3	Cortex API to get the result of a job is invoked and the response is analyzed to check if there is a report and success is True.		
Input data	{"si-em_label":"TA0010","containment":true,"coas":[{"defenses":[{"defenseName":"Host isolation","defenseInfo":"Isolate the host from the network by using Network Segmenta-tion","mitreRef":"M1030","hosts":["10.255.130.79","10.255.131.163"]},{"defenseName":"Trafic filtering","defenseInfo":"Use network appliances to filter ingress or egress traffic and perform protocol-based filter-ing","mitreRef":"M1037","flows":[{"src":"10.255.131.163","dst":"10.255.130.79"}]}]}]}		
	Ranked CoAs		

Expected result	Ranked CoAs
	
	{
	"summary": {},
	"full": {
	"ranked_coas": [
	{
	"RORI": null,
	"business_impact": 0.4224,
	"containment": true,
	"playbook": {
	"type": "playbook",
	"spec_version": "1.0",
	"created_by": "Soccrates_Response_Planner",
	"description": "This playbook will apply the CoA in the system",
	"playbook_types": [
	"prevention"
	],
	"features": {
	"parallel_processing": true
	},
	"created": "Mon Feb 28 09:50:39 2022",
	"id": "soccrates-playbook-1646041839718883200",
	"name": "Soccrates CoA Playbook 1646041839718883200",
	"workflow_start": "start--soccrates-playbook-1646041839718883200",
	"workflow": {
	"start--soccrates-playbook-1646041839718883200": {
	"type": "start",
	"on_completion": "parallel--soccrates-playbook-1646041839718883200"
	},
	"parallel--soccrates-playbook-1646041839718883200": {
	"type": "parallel",
	"name": "Apply defenses of the CoA",
	"next_steps": [
	"defense-1--soccrates-playbook-1646041839718883200",
	"defense-2--soccrates-playbook-1646041839718883200"
	]
	},
	"defense-1--soccrates-playbook-1646041839718883200": {
	"type": "single",
	"name": "Host isolation",
	"description": "Isolate the host from the network by using Network Segmentation",
	"on_completion": "end--soccrates-playbook-1646041839718883200",
	"target": {
	"type": "individual",
	"email": {

```
        "work": "human.operator@default.email"
       }
      },
      "commands": [
       {
        "type": "manual",
        "command": "Isolate the host from the network by using Network Segmentation on these assets: ['10.255.130.79', '10.255.131.163']"
       }
      ]
     },
     "defense-2--soccrates-playbook-1646041839718883200": {
      "type": "single",
      "name": "Trafic filtering",
      "description": "Use network appliances to filter ingress or egress traffic and perform protocol-based filtering",
      "on_completion": "end--soccrates-playbook-1646041839718883200",
      "target": {
       "type": "individual",
       "email": {
        "work": "human.operator@default.email"
       }
      },
      "commands": [
       {
        "type": "manual",
        "command": "Use network appliances to filter ingress or egress traffic and perform protocol-based filtering on these assets: [{'src': '10.255.131.163', 'dst': '10.255.130.79'}]"
       }
      ]
     },
     "end--soccrates-playbook-1646041839718883200": {
      "type": "end"
     }
    }
   },
   "defenses": [
    {
     "defenseName": "Host isolation",
     "defenseInfo": "Isolate the host from the network by using Network Segmentation",
     "mitreRef": "M1030",
     "hosts": [
      "10.255.130.79",
      "10.255.131.163"
     ],
```


Classification level: Public

	"ref": [
	"10.255.130.79",
	"10.255.131.163"
	]
	},
	{
	"defenseName": "Trafic filtering",
	"defenseInfo": "Use network appliances to filter ingress or egress traffic and perform protocol-based filtering",
	"mitreRef": "M1037",
	"flows": [
	{
	"src": "10.255.131.163",
	"dst": "10.255.130.79"
	}
	],
	"ref": [
	"10.255.131.163",
	"10.255.130.79"
	]
	}
	],
	"monetary_cost": null,
	"defenses_hosts": [
	"10.255.131.163",
	"10.255.130.79"
	]
	}
	],
	"report_url": "http://10.8.0.1:8001/?request=1646041839722478776"
	},
	"success": true,
	"artifacts": [],
	"operations": []
	}

Test Case Result	Success

Test Case ID	C.10	Components	
Type of testing	Integration	Orchestrator	SOCCRATES_RP_CONTAINMENT_COA_GENERATOR_1_0 (Response Planner)
Description	Containment CoA Generator Cortex Analyzer integration with Response Planner (preliminary response of UC1)		
Specification	D2.3 Figure 4.6 [1]	Priority	Low/Medium/High
Prepared by	Susana González	Tested by	Susana González
Pre-condition(s)			
Test steps			
1	Cortex API to run an analyzer is invoked from pytest script with the id of the SOCCRATES_RP_CONTAINMENT_COA_GENERATOR Analyzer and the request data.		
2	The job id of the execution is retrieved and it is waited a timeout for a response.		
3	Cortex API to get the result of a job is invoked and the response is analyzed to check if there is a report and success is True.		
Input data	{"type":"stop_exfiltration","siem_label":"TA0010","sourceIPs":["10.255.129.220"],"destinationIPs":["10.255.131.49"]}		

Expected result	{
	"summary": {},
	"full": {
	"containment": true,
	"coas": [
	{
	"defenses": [
	{
	"defenseName": "Traffic filtering",
	"defenseInfo": "Use network appliances to filter ingress or egress traffic and perform protocol-based filtering",
	"flows": [
	{
	"src": "10.255.129.220",
	"dst": "10.255.131.49"
	}
	],
	"mitreRef": "M1037"
	}		

	] }] }, "success": true, "artifacts": [], "operations": [] }
Test Case Result	Success

Test Case ID	C.11	**Components**	
Type of testing	Integration	**Orchestrator**	SOCCRATES_RP_CONTAINMENT_COA_GENERATOR_1_0 (Response Planner)
Description	Containment CoA Generator Cortex Analyzer integration with Response Planner (final containment of UC1)		
Specification	D2.3 Figure 4.9 [1]	**Priority**	Low/Medium/High
Prepared by	Susana González	**Tested by**	Susana González
Pre-condition(s)			
Test steps			
1	Cortex API to run an analyzer is invoked from pytest script with the id of the SOCCRATES_RP_CONTAINMENT_COA_GENERATOR Analyzer and the request data.		
2	The job id of the execution is retrieved and it is waited a timeout for a response.		
3	Cortex API to get the result of a job is invoked and the response is analyzed to check if there is a report and success is True.		
Input data	{"type":"containment","siem_label":"TA0010","sourceIPs":["10.255.129.220"],"destinationIPs":["10.255.131.49"]}		

Expected result	{
	"summary": {},
	"full": {
	"containment": true,
	"coas": [		

	{
	"defenses": [
	{
	"defenseName": "Host isolation",
	"defenseInfo": "Isolate the host from the network by using Network Segmentation",
	"hosts": [
	"10.255.129.220",
	"10.255.131.49"
	],
	"mitreRef": "M1030"
	}
	]
	}
	]
	},
	"success": true,
	"artifacts": [],
	"operations": []
	}

Test Case Result	Success

Test Case ID	C.12	**Components**	
Type of testing	Integration	**Orchestrator**	SOCCRATES_Coa_Generator_1_0 (CoA Generator)
Description	CoA Generator Cortex Analyzer integration with CoA Generator		
Specification	D2.3 Figure 4.11 [1]	**Priority**	Low/Medium/High
Prepared by	Susana González	**Tested by**	Susana González
Pre-condition(s)	The id included in the request data must correspond to an existent securiCAD simulation id generated from the existing IT infrastructure		
Test steps			
1	Cortex API to run an analyzer is invoked from pytest script with the id of the CoA Generator Cortex Analyzer and the request data.		
2	The job id of the execution is retrieved and it is waited a timeout for a response.		

3	Cortex API to get the result of a job is invoked and the response is analyzed to check if there is a report and success is True.
Input data	{\"simulationId\":\"307666084811239\"}
Expected result	List of CoAs


```
{
 "summary": {},
 "full": {
  "CoAs": [
   {
    "monetary_cost": {
     "1": 2260
    },
    "defenses": [
     {
      "ref": "cea7f972-55ce-4233-a8f6-e190e4e19e3b",
      "defenseName": "Restricted",
      "defenseInfo": "Restricted is used"
     }
    ],
    "coaTTC": {
     "48639a8b-8fcd-4802-af3f-f8f28cd24901.Deny": [
      1.7976931348623157e+308,
      1.7976931348623157e+308,
      1.7976931348623157e+308
     ],
     "696eef6c-f4f1-4aed-a4cf-a9b503accd1f.Modify": [
      0,
      1.7976931348623157e+308,
      1.7976931348623157e+308
     ],
     "1ee379f5-24a6-496a-b88a-0b424dbcc394.AttemptAssume": [
      1.7976931348623157e+308,
      1.7976931348623157e+308,
```

```
      1.7976931348623157e+308
     ]
    },
    "report_url": "https://34.91.238.96/project/446664605265379/scenario/1539
17211115599/report/338066619188934",
    "efficiency": "149.993"
   },
   {
    "monetary_cost": {
     "1": 2242
    },
    "defenses": [
     {
      "ref": "cea7f972-55ce-4233-a8f6-e190e4e19e3b",
      "defenseName": "Restricted",
      "defenseInfo": "Restricted is used"
     },
     {
      "ref": "23d01b06-7402-4c84-b0af-af2418e2ef09",
      "defenseName": "Remove",
      "defenseInfo": "Remove is used"
     }
    ],
    "coaTTC": {
     "48639a8b-8fcd-4802-af3f-f8f28cd24901.Deny": [
      1.7976931348623157e+308,
      1.7976931348623157e+308,
      1.7976931348623157e+308
     ],
     "696eef6c-f4f1-4aed-a4cf-a9b503accd1f.Modify": [
      1.7976931348623157e+308,
      1.7976931348623157e+308,
      1.7976931348623157e+308
```

```
    ],
    "1ee379f5-24a6-496a-b88a-0b424dbcc394.AttemptAssume": [
     1.7976931348623157e+308,
     1.7976931348623157e+308,
     1.7976931348623157e+308
    ]
   },
   "report_url": "https://34.91.238.96/project/446664605265379/scenario/1539
17211115599/report/277440937705481",
   "efficiency": "149.993"
  }
 ],
 "initialTTC": {
  "48639a8b-8fcd-4802-af3f-f8f28cd24901.Deny": [
   0,
   1.7976931348623157e+308,
   1.7976931348623157e+308
  ],
  "696eef6c-f4f1-4aed-a4cf-a9b503accd1f.Modify": [
   0,
   1.7976931348623157e+308,
   1.7976931348623157e+308
  ],
  "1ee379f5-24a6-496a-b88a-0b424dbcc394.AttemptAssume": [
   1.7976931348623157e+308,
   1.7976931348623157e+308,
   1.7976931348623157e+308
  ]
 }
},
"success": true,
"artifacts": [],
"operations": []
```

	}
Test Case Result	Success

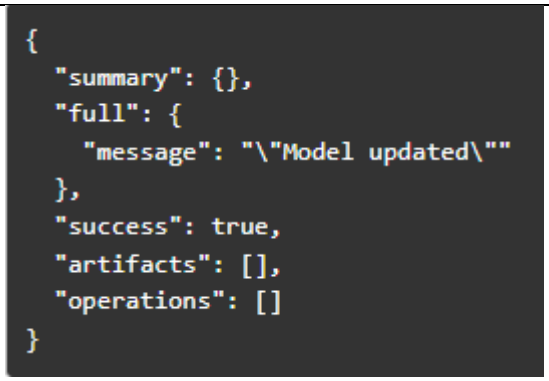
Test Case ID	C.13	**Components**	
Type of testing	Integration	**Orchestrator**	SOCCRATES_AAD_RESPONDER_1_0
Description	AAD Cortex Responder integration with AI based Attack Detection component to update AAD model with new asset.		
Specification	D2.3 Figure 4.11 [1]	**Priority**	Low/Medium/High
Prepared by	Susana González	**Tested by**	
Pre-condition(s)	Cortex and AAD server are up and running		
Test steps			
1	`Cortex API to run an analyzer is invoked from pytest script with the id of the AAD Cortex Responder and the request data.`		
2			
3			
Input data	`{"timestamp":"2022-02-21T16:45:14Z","type":"newAsset","host":"atos-test-21-02-2022_1"}`		

Expected result	{
	"summary": {},
	"full": {
	"message": "Successfully sent event to Reasens"
	},
	"success": true,
	"artifacts": [],
	"operations": []
	}		

Test Case Result	Success		

Test Case ID	C.14	Components	
Type of testing	Integration	Orchestrator	SOCCRATES_AAD_RESPONDER_1_0
Description	AAD Cortex Responder integration with AI based Attack Detection component to update AAD model with new vulnerability.		
Specification	D2.3 Figure 4.11 [1]	Priority	Low/Medium/High
Prepared by	Susana González	Tested by	
Pre-condition(s)	Cortex and AAD server are up and running		
Test steps			
1	Cortex API to run an analyzer is invoked from pytest script with the id of the AAD Cortex Responders and the request data		
2			
3			
Input data	{"timestamp":"2020-12-12T16:45:14Z","type":"newVulnerability","host":"computer01.soccrates.eu","vulnerabilities":[{"software":"cpe:2.3:a:google:chrome:*:*:*:*:*:*:*:*","cve":"CVE-2021-21112"}]}		
Expected result	{ "summary": {}, "full": { "message": "Successfully sent event to Reasens" }, "success": true, "artifacts": [], "operations": [] }		
Test Case Result	Success		

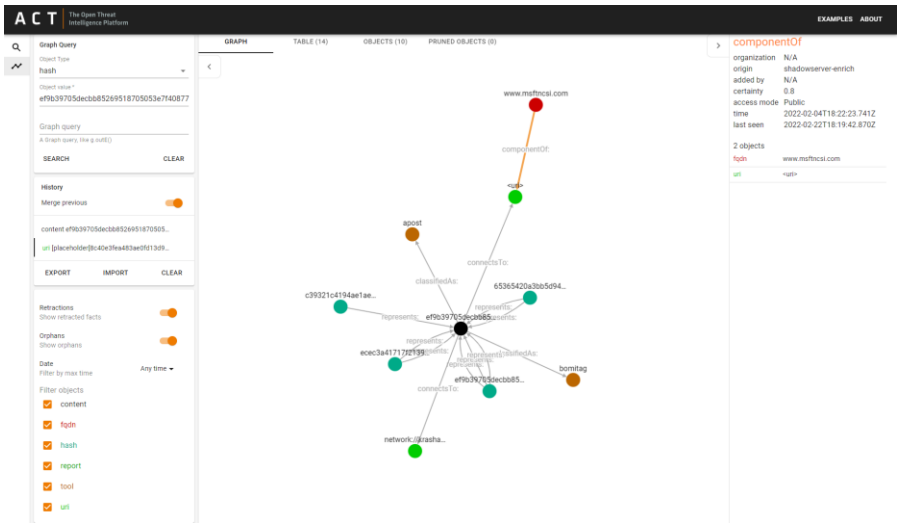
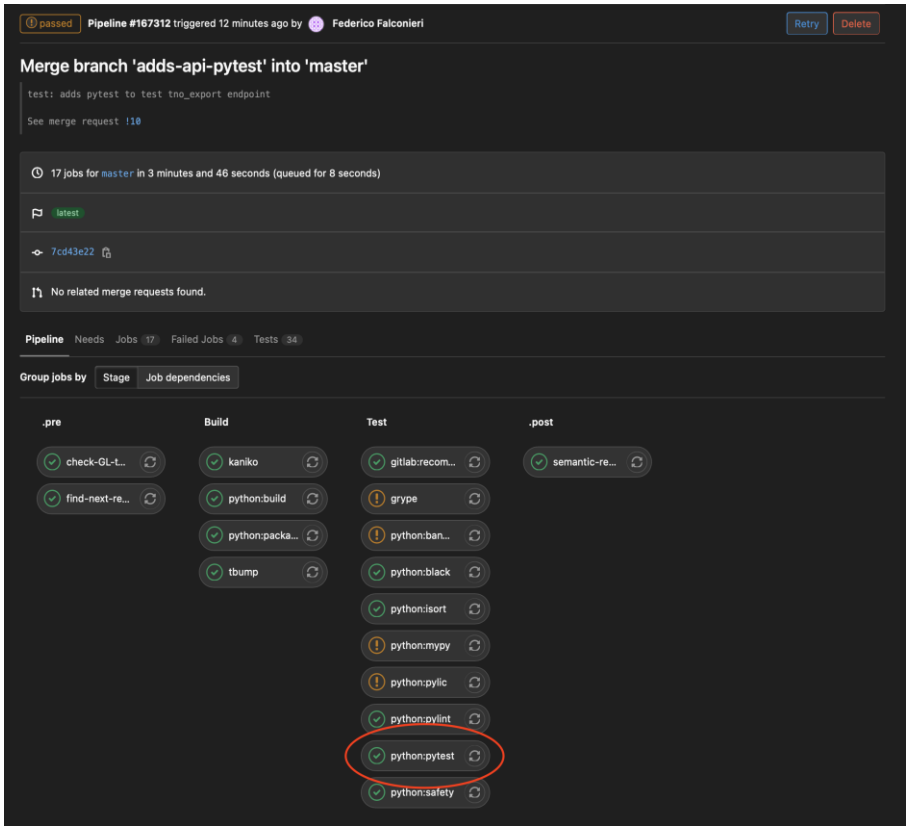
Test Case ID	C.15	Components	
Type of testing	Integration	Orchestrator	SOCCRATES_Business_Impact_responder _1_0
Description	BIA Cortex Responder integration with BIA		
Specification	D2.3 Figure 4.11 [1]	Priority	Low/Medium/High

Prepared by	Susana González	**Tested by**	Susana González
Pre-condition(s)	UC5 has been triggered in the orchestrator		
Test steps			
1	Cortex API to run an analyzer is invoked from pytest script with the id of the AAD Cortex Responders and the request data		
2			
3			
Input data	{"newAsset":["atos-test-21-02-2022_1"]}		
Expected result			
Test Case Result	Success		

Test Case ID	C.16	**Components**	
Type of testing	Integration	**Orchestrator**	Mailer_1_0 (Automated Response)
Description	AR Mailer Cortex Responder integration with SMTP server to notify the execution of selected CoAs		
Specification	D2.3 Figure4.6, Figure 4.9, Figure 4.11 [1]	**Priority**	Low/Medium/High
Prepared by	Susana González	**Tested by**	Susana González
Pre-condition(s)	Some of the use cases (UC1-5) has been triggered in the orchestrator		
Test steps			
1	Cortex API to run an analyzer is invoked from pytest script with the id of the AAD Cortex Responders and the request data		
2	Check email is received		

3	
Input data	{"description":" Response Planner Containment CoAs to stop ex-filtration: \\\\n\\\\n\\\\tContainment CoA Defense: \\\\n\\\\t\\\\t - Host isolation(M1030): Isolate the host from the network by using Network Segmentation (hosts: [10.255.129.220, 10.255.131.49])\\\\n\\\\t\\\\t - Trafic fil-tering(M1037): Use network appliances to filter ingress or egress traffic and perform protocol-based filtering (flows: [{src=10.255.129.220, dst=10.255.131.49}])\\\\n\\\\nTriggering Message: {\\\\\\\\"SourceIP\\\\\\\\": \\\\\\\\"10.255.129.220\\\\\\\\", \\\\\\\\"SourcePort\\\\\\\\": \\\\\\\\"1234\\\\\\\\", \\\\\\\\"DestinationIP\\\\\\\\": \\\\\\\\"10.255.131.49\\\\\\\\", \\\\\\\\"DestinationPort\\\\\\\\": \\\\\\\\"80\\\\\\\\", \\\\\\\\"Severity\\\\\\\\": \\\\\\\\"High\\\\\\\\", \\\\\\\\"Tactic\\\\\\\\": \\\\\\\\"TA0010\\\\\\\\", \\\\\\\\"Date\\\\\\\\": \\\\\\\\"2021-12-02T11:38:05.098Z\\\\\\\\ }","title":"CoAs suggested by SOCCRATES platform as Response on Preliminary response (exfiltration) when detected ongoing at-tack (UC-1)","tags":["mail:soccrates.man-ager@gmail.com"],"username":"soccrates.manager@gmail.com"}
Expected result	{ "summary": {}, "full": { "message": "message sent" }, "success": true, "artifacts": [], "operations": [{ "type": "AddTagToCase", "tag": "mail sent" }] }
Test Case Result	Success

Test Case ID	C.17	Components	
Type of testing	Integration	TIP	Threat Data Collection & Threat Prediction
Description	TIP integration with the Shadowserver API to enrich newly received CTI.		
Specification	D2.3 Table 3.24 [1]	Priority	Low/~~Medium/High~~
Prepared by	Martin Eian	Tested by	Fredrik Borg
Pre-condition(s)	TIP and Shadowserver API are up and running. TIP has Internet access.		
Test steps			
1	Trigger enrichment of a file hash in the TIP		
2	Observe output of the TIP shadowserver-enrich worker		
3	Access the TIP GUI, search for the file hash, verify that new facts with origin « shadowserver-enrich » have been added.		
Input data	echo ef9b39705decbb85269518705053e7f4087758eea6bab4ba9135bf1ae922b2ea | act-shadowserver-enrich		
Expected result	Facts of type « represents », « connectsTo » and « componentOf » added.		
Test Case Result	Success		



Test Case ID	C.18	Components	
Type of testing	Integration	**AAD – middleman**	Threat Data Collection & Threat Prediction
Description	Middleman integration with the Shadowserver API to collect latest list of all domains classified as DGAs from the past day.		
Specification	D2.3 Table …	**Priority**	Low/Medium/High
Prepared by	Federico Falconeri	**Tested by**	Federico Falconeri
Pre-condition(s)	Middleman code on gitlab and Shadowserver API is up and running.		
Test steps			
1	`Trigger a CI pipeline in the middleman repository (either manually or automatically when there are new commits)`		
2	`CI will run a pytest unit test that will retrieve data from the research/tno-export endpoint of shadowserver API`		
3	`The test checks that the endpoint returns a list of domains larger than 0`		
Input data	N/A		
Expected result	The endpoint returns more than 0 domains		
Test Case Result	Success