D5.4 Appendix J

ATE – Tests – Vertical 2 – SAML IdP Server Scenario

Project number	830892
Project acronym	SPARTA
Project title	Strategic programs for advanced research and technology in Europe
Start date of the project	1st February, 2019
Duration	36 months
Programme	H2020-SU-ICT-2018-2020

Deliverable type	Report
Deliverable reference number	SU-ICT-03-830892 / D5.4/ v1.0 / Appendix J
Work package contributing to the deliverable	WP5
Due date	Jan 2022 – M36
Actual submission date	2nd February, 2022

Responsible organisation	SAP
Editor	Henrik Plate
Dissemination level	PU
Revision	V1.0

Abstract	This document provides a description of the test procedure and report for the SAML IdP Server scenario of the "Complex System Assessment Including Large Software and Open-Source Environments, targeting e-Government Services" vertical (also known as e-Government services vertical or Vertical 2). The document demonstrates how the CAPE tools ProjectKB, Steady and VI contribute to secure the SAML IdP Server scenario.
Keywords	Vulnerability Assessment, Security Tests

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 830892.

Editor

Henrik Plate (SAP)

Contributors

Mirko Malacario, Claudio Porretti (LEO)

Reviewers

Maximilian Tschirschnitz (TUM)

Rimantas Zylius (L3CE)

Table of Content

List of Figures

List of Tables

Chapter 1 Introduction

1.1 Document Overview

This document provides a description of the test procedure and report for the SAML IdP Server Scenario (also known as CIE ID SERVER) of the “Complex System Assessment Including Large Software and Open-Source Environments, Targeting e-Government Services” vertical (a.k.a. e-Government services vertical or Vertical 2).

We will show how the CAPE tools Steady, ProjectKB and VulnEx contribute to secure the SAML IdP Server scenario. In particular, we will show how:

- we properly integrated in the development process of the SAML IdP Server the continuous integration techniques developed in the context of Task 5.3 for Steady, ProjectKB and VulnEx, and
- Steady, ProjectKB and VulnEx perform a security assessment of the SAML IdP Server, providing a security report to the security analyst.

The structure of the document is organized as follows:

- Chapter 1 Introduction, is the current section presenting the objectives, scope and structure of the document.
- Chapter 2 Test preparations, presents the hardware and software used for testing.
- Chapter 3 Test descriptions, details the different test cases to be executed and their results.
- Chapter 4 Test Summary Coverage, shows the completeness of tests coverage.

Chapter 2 Test preparations

2.1 *System overview*

The e-Government services vertical (Vertical 2) has been fully described in D5.2 [1]. In this section, we provide an overview of the case study description, focusing on the SAML IdP Server scenario.

The demonstration scenario of the vertical 2 involves the development and testing environments managed by FBK (one of the institutions of the SPARTA partner CINI), where the preliminary versions of the SAML IdP Server is developed, deployed, and tested, before being migrated on the Italian Ministry of the Interior servers.

The SAML IdP Server is developed by the Italian National Mint and Printing House. It is a custom implementation based on Shibboleth, a standard-based, open-source software package for Single Sign-on (SSO) system across or within organizational boundaries. SAML IdP is responsible for supplying information about users at a domain to relying parties protected by service providers leveraging the information contained in the Italian electronic identity card (CIE 3.0).

Each communication is through an API REST via HTTPS. Shibboleth is mostly a set of software components made using the Spring framework based on Java programming language and built with Apache Maven.

As mentioned in D5.3 [2], FBK has extended the Gitlab environment in such a way to use the continuous integration functionalities offered by Gitlab. Every time a git commit is pushed on the repository, the source code is automatically built and deployed (using Apache Maven) on an Azure virtual machine.

The system under test used for SPARTA consists of the source code of the SAML IdP Server and its integration in the DevSecOps pipeline.

To avoid any risk of disclosing sensitive information concerning the official version of the SAML IdP Server, the tests are performed on an old version of the source code of the server. Indeed, the purpose of the tests is to show that the CAPE tools are properly integrated in the development process and are indeed helpful to spot relevant vulnerabilities.

2.1.1 Hardware preparation

The dockerized versions of Steady, ProjectKB and VulnEx runs on a virtual machine (VM), so there is no needed hardware preparation for using the tools. The VM used to host the tools is an Azure Standard DS3 v2 equipped with a processor Intel Xeon E5-2673 v4 2.29 GHz 4 Cores, 14 GB of RAM and two drives, one hard disk drive of 30 GB used by the OS and 10GB of solid-state drive used by the tools.

2.1.2 Software preparation

The testing environment consists of a Gitlab platform hosted by FBK, and cloud-hosted Azure virtual machines controlled by FBK.

Gitlab provides:

- a version control system (Git-repository), storing the source code of SAML IdP Server;
- Issues tracking and continuous integration and deployment pipeline.

The Azure VM hosting the CAPE tools runs Linux distributions (Ubuntu 20.04) and supports the Docker technology. We installed in the VM the GitLab Runner application, which works with GitLab CI/CD to run jobs in a pipeline.

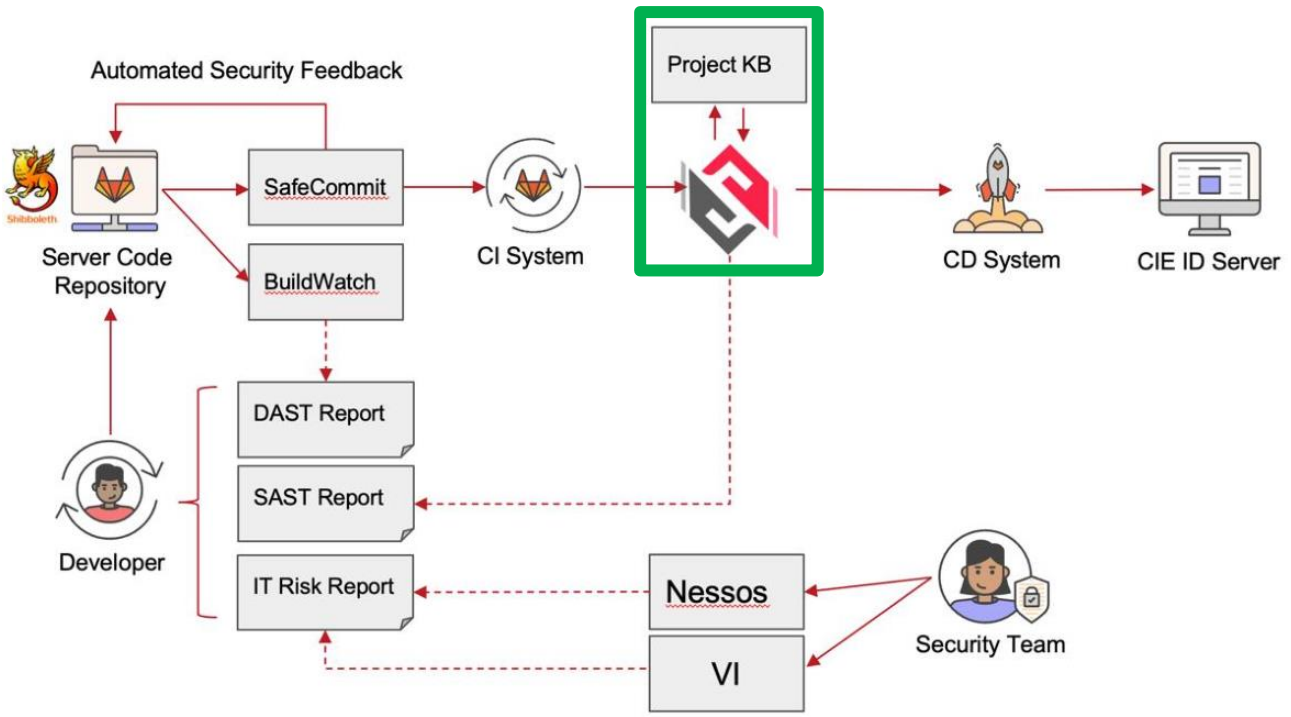

Figure 1: Build pipeline with CAPE tools

As mentioned in D5.3, to assess the security of the SAML IdP Server, we deployed the DevSecOps scenario depicted in Figure 1. In particular, concerning ATE, the Steady and ProjectKB CAPE tools are used to evaluate the security and risk requirements for the SAML IdP Server (highlighted in green). The pipeline has been designed as a set of integration scripts that are attached to the GitLab repository hosting the SAML IdP Server source code.

Chapter 3 Test descriptions

Table 1 shows the Security Requirements (SRs) allocated to Steady/VI in the traceability matrix for Vertical 2 as per D5.4 [3] for the SAML IdP Server. In the next sections we describe the test descriptions that have been elaborated to support the test of these requirements.

Security Req (ID)	Short Description
T1195.001-S1	The SAML IdP Server must not depend on components with known vulnerabilities. (This can be checked in the vulns and mitigation tabs).
T1195.001-S2	The SAML IdP Server must not depend on components that are unmaintained or do not produce security patches anymore. (This can be checked in the dependencies and mitigation tabs)
T1195.001-S3	The SAML IdP Server must not include un-used components. (This can be checked through Steady's reachability analysis, displayed in the dependencies tab)

Table 1: Security Requirements covered by Steady, ProjectKB and VulnEx

3.1 *T1195.001-S1_TC1*

3.1.1 Security Requirements addressed

T1195.001-S1

The use of components with known vulnerabilities is among the OWASP Top 10 security risks for Web applications. Such vulnerable components represent low hanging fruits for attackers, since exploits are readily available and can be easily and automatically tested on internet-facing applications.

The corresponding requirements are as follows:

1) It is required to check on a continuous basis whether the application depends on components with known vulnerabilities.
2) Every component vulnerability discovered (finding) requires an assessment regarding (a) its actual exploitability in the context of the application, and (b) the potential impact on its integrity, confidentiality and availability requirements.
3) The assessment must be reflected either by updating the respective component(s) (such that the finding disappears), or by providing a rationale why such update is not necessary (e.g. because of non-exploitability, because the components in question are test dependencies, or because safeguards are implemented by the application, etc.).
4) Whenever component updates are not performed, the reasoning must be documented in the versioning control system, for tracking purposes and to make the exemption available to the scan tool.

3.1.2 Test preconditions

- An instance of Steady's backend application is reachable from the CI/CD system.
- The database of this instance has been populated with vulnerability information loaded from Project KB.
- The CI/CD system is able to download the scan client needed for the respective development project from Maven Central (Steady's plugin for Maven is required for the SAML IdP Server).

- The build job is configured to invoke Steady's scan client on the given development project such that the build job breaks in case of non-exempted (non-justified) component vulnerabilities. The detailed analysis goals to be configured in the build job are as follows:
  - The goal APP creates a method-level bill-of-materials of the application and all its dependencies, and identifies all component vulnerabilities.
  - The goals A2C, PREPARE-AGENT, TEST and UPLOAD collect information regarding the reachability of vulnerable code, thus, input supporting the above-mentioned risk assessment.
  - The goal REPORT creates a summary report in different formats and breaks the build job depending on the findings.
- The Steady <workspace> used for persisting the analysis results is known to the evaluator.

3.1.3 Expected test results

Build jobs must succeed, which means that either there are no known component vulnerabilities or they are properly exempted/justified.

3.1.4 Criteria for evaluating results

The only criteria is whether the build succeeds or not. If it fails, the application under tests violates the requirements mentioned before.

3.1.5 Test Procedure

The following steps must be followed in order to obtain the test results:

- Point your browser to the GitLab Web frontend and open the respective/latest build job
- In case the build job failed because of Steady's REPORT goal, one can find the details about known vulnerabilities with help of two alternative steps:
  - Open the context menu of the build job → Download and extract the ZIP file containing Steady's result reports → Open the HTML report (cf. screenshot in Figure 3) and document all vulnerabilities
  - Point your browser to the Steady Web frontend (http://<host>:8033/apps/#/<workspace>) → Select all modules of the project under analysis (using the table on the left-hand side) → For each module: open the tab "Vulnerabilities" on the right-hand side (cf. example screenshot in Figure 2) → Document all vulnerabilities (archive, CVE) that have not been exempted (i.e. have no audit icon in the left-most column)

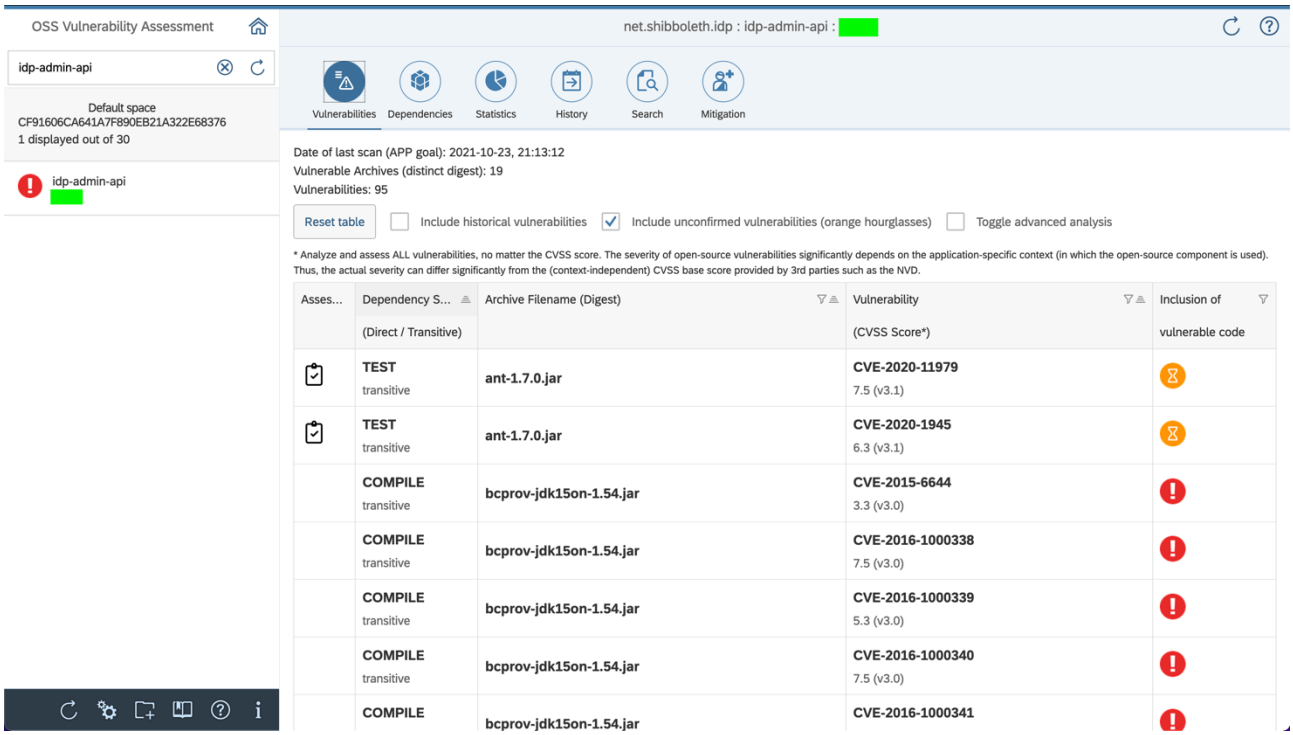

Figure 2: Example screenshot of the "Vulnerabilities" tab in Steady's Web frontend

3.1.6 Test Results

Status: FAILED

The Html report, created by Steady's REPORT goal, and a screenshot of which is depicted below, provides the following information:

- The analysis has been performed on Nov 11, 2021 with Eclipse Steady v3.2.0.
- The analysis concluded with a failure, due to the presence of 75 vulnerabilities. Those exist in 15 different Java components, as visible in the Table below.
- 22 other vulnerabilities have been exempted due to Steady's default configuration (which exempts vulnerabilities in dependencies of type test and provided).

The screenshot also visualizes more detailed information for one of the vulnerabilities, CVE-2016-1000338 in Java component bcprov-jdk15on-1.54.jar. The vulnerability has a CVSS rating of 7.5, and – illustrated by red paws - the vulnerable code is potentially reachable in module idp-authn-impl (and others), according to static source code analysis.

The different links in the report lead to Steady's user frontend, where more information is provided.

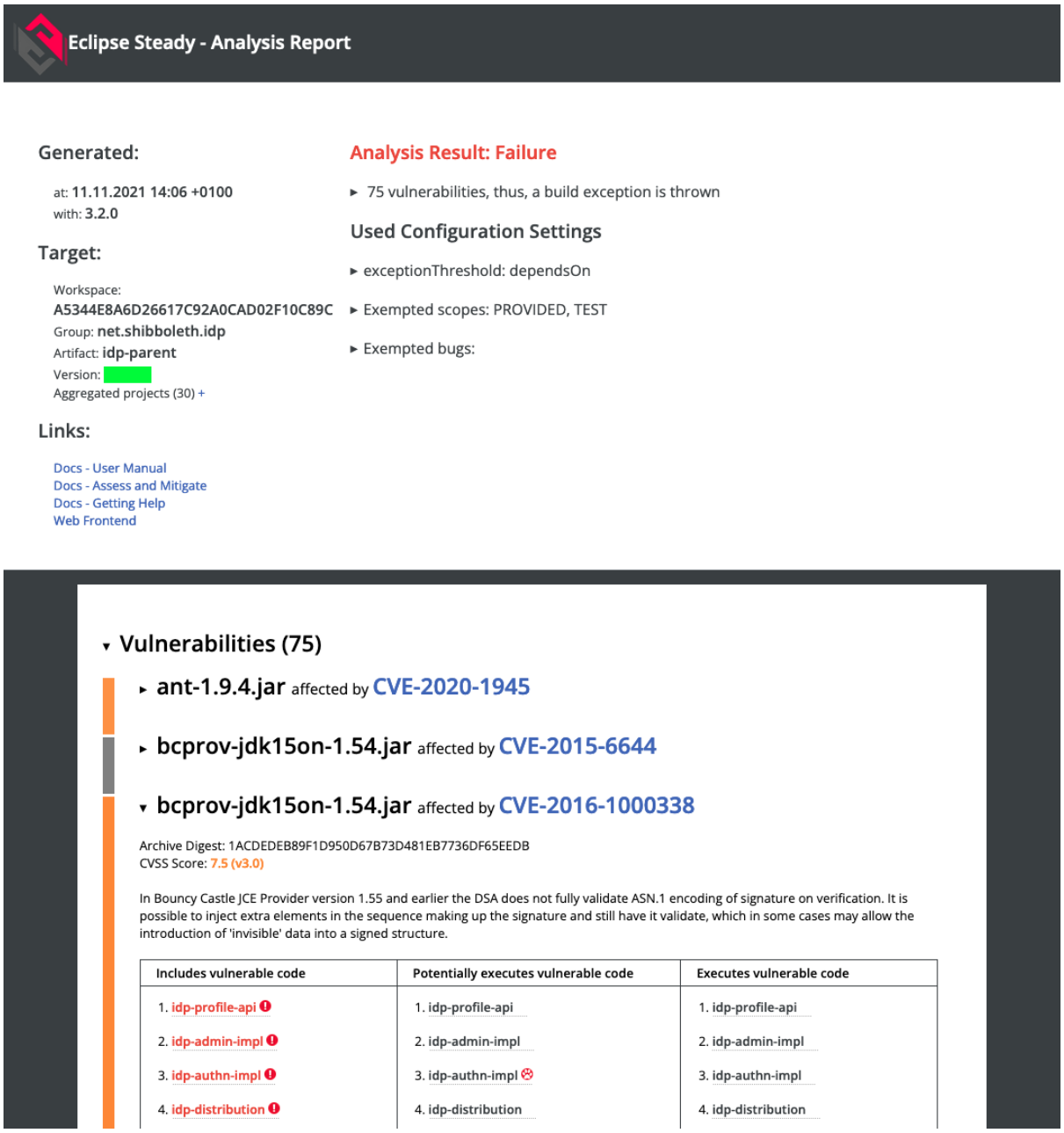

Figure 3: Screenshot of Steady's Html result report

The following Table 2contains all component vulnerabilities identified in the project under analysis (across all 30 modules):

#	Component	CVE
1	ant-1.9.4.jar	CVE-2020-1945
2	bcprov-jdk15on-1.54.jar	CVE-2015-6644
3		CVE-2016-1000338
4		CVE-2016-1000339
5		CVE-2016-1000340
6		CVE-2016-1000341
7		CVE-2016-1000342

#	Component	CVE
8		CVE-2016-1000343
9		CVE-2016-1000344
10		CVE-2016-1000345
11		CVE-2016-1000346
12		CVE-2016-1000352
13		CVE-2018-1000180
14		CVE-2018-1000613
15		CVE-2019-17359
16	c3p0-0.9.2.1.jar	CVE-2018-20433
17		CVE-2019-5427
18	cryptacular-1.1.1.jar	CVE-2020-7226
19	dom4j-1.6.1.jar	CVE-2018-1000632
20	guava-19.0.jar	CVE-2018-10237
21	jackson-databind-2.8.3.jar	CVE-2017-15095
22		CVE-2017-17485
23		CVE-2017-7525
24		CVE-2018-11307
25		CVE-2018-12022
26		CVE-2018-12023
27		CVE-2018-14718
28		CVE-2018-14719
29		CVE-2018-14720
30		CVE-2018-14721
31		CVE-2018-19360
32		CVE-2018-19361
33		CVE-2018-19362
34		CVE-2018-5968
35		CVE-2018-7489
36		CVE-2019-12086
37		CVE-2019-12384
38		CVE-2019-12814
39		CVE-2019-14379
40		CVE-2019-14439
41		CVE-2019-14540
42		CVE-2019-14892
43		CVE-2019-14893

#	Component	CVE
44		CVE-2019-16942
45		CVE-2019-16943
46		CVE-2019-17267
47		CVE-2019-17531
48		CVE-2019-20330
49		CVE-2020-10650
50		CVE-2020-10672
51		CVE-2020-10673
52		CVE-2020-10968
53		CVE-2020-10969
54		CVE-2020-11111
55		CVE-2020-11112
56		CVE-2020-11113
57		CVE-2020-24616
58		CVE-2020-24750
59		CVE-2020-8840
60		CVE-2020-9546
61		CVE-2020-9547
62		CVE-2020-9548
63	logback-classic-1.1.3.jar	CVE-2017-5929
64	logback-core-1.1.3.jar	CVE-2017-5929
65	retrofit-2.1.0.jar	CVE-2018-1000850
66	spring-core-4.3.2.RELEASE.jar	CVE-2018-1272
67	spring-expression-4.3.2.RELEASE.jar	CVE-2018-1270
68		CVE-2018-1275
69	spring-web-4.3.2.RELEASE.jar	CVE-2018-15756
70		CVE-2020-5421
71	spring-webflow-2.4.4.RELEASE.jar	CVE-2017-4971
72		CVE-2017-8039
73	spring-webmvc-4.3.2.RELEASE.jar	CVE-2016-9878
74		CVE-2018-1271
75		CVE-2020-5421

Table 2: Application dependencies subject to known vulnerabilities

3.1.6.1 Deviations from test procedure

None

3.2 *T1195.001-S2_TC1*

3.2.1 Security Requirements addressed

T1195.001-S2

The regular update of components to recent releases, independent of known vulnerabilities, is important for several reasons:

First, in case a vulnerability becomes disclosed, an update to a non-vulnerable version is easier if the gap between the version in use and the target version is small. Secondly, old components risk to run out of maintenance, thus, no fixes will be produced anymore, and nobody validates whether newly discovered vulnerabilities affect such old components. Finally, using later releases can also mean to benefit from hidden security fixes, which have not been publicly communicated by the open-source project.

Besides regular updates, it is preferable to avoid custom-built versions of open-source components, e.g. forks of the official open-source project, or otherwise modified versions. The reason is that the security fix of custom versions or forks can be significantly more difficult than updating to non-vulnerable versions of the standard component.

The corresponding requirements are as follows:

- The application must not depend on components that are older than 60 months (5 years).
- The application must not depend on components having digests that are unknown to Maven Central.

3.2.2 Test preconditions

- A build job ran Steady's APP analysis goal on the project under analysis.
- The Steady <workspace> used for persisting the analysis results is known to the evaluator.

3.2.3 Expected test results

None of the dependencies has been released more than 5 years ago.

All of the dependencies have a SHA1 digest known to Maven Central (unless the dependency is a module of the same development project, thus, has been created in the context of the same build job).

3.2.4 Criteria for evaluating results

No outdated and non-standard components in the different modules of the project under analysis.

3.2.5 Test Procedure

- Point your browser to the Steady Web frontend (http://<host>:8033/apps/#/<workspace>)
- Select all modules of the project under analysis (using the table on the left-hand side)
- For each module: open the tab "Dependencies" on the right-hand side (cf. example screenshot in Figure 4)
- Document all dependencies with "False" in column "Well-known digest", and all dependencies with a release date older than 5 years in column "Release date"

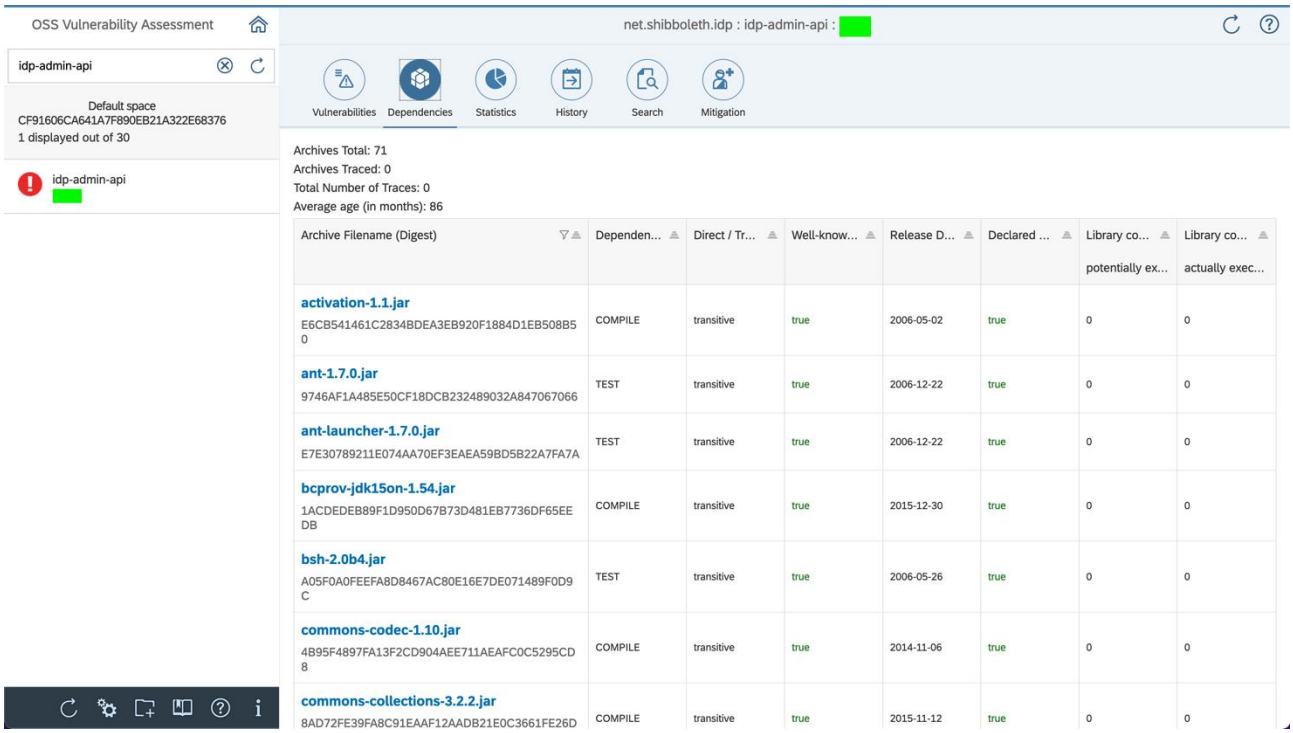

Figure 4: Example screenshot of the "Dependencies" tab in Steady's Web frontend

3.2.6 Test Results

Status: FAILED

The following table contains 10 dependencies of the project under analysis (out of 147 distinct dependencies across all 30 project modules) with SHA1 digests unknown to Maven Central.

Note, however, that a manual check of the digest of javassist-3.18.1-GA.jar revealed that the archive is indeed present in Maven Central. This false-positive is due to a bug in Maven Central's search API[1].

#	Filename of dependency	SHA1 digest	Scope
1	opensaml-saml-api-3.4.0.jar	538E1E54E5E8160F2D284B08F1B8A7B93053E0DA	COMPILE
2	opensaml-saml-impl-3.4.0.jar	06336645EC0B0FBD98A7A5E719B4C4C284A4D79F	COMPILE
3	opensaml-security-impl-3.4.0.jar	EE6158D53B576D6A63D3C7A0CF063C8518E75126	COMPILE
4	opensaml-soap-api-3.4.0.jar	830B14C47A7E3E21ED377BE4C82E6F19FF5C6749	COMPILE
5	opensaml-soap-impl-3.4.0.jar	23F0B2732C87A34C0179584E71A52839EAF9C186	COMPILE
6	opensaml-xmlsec-impl-3.4.0.jar	94EA339D9E63436CDF4A2247B5EF85867E66D302	COMPILE

[1] Reproducible via https://search.maven.org/search?q=1:D9A09F7732226AF26BF99F19E2CFFE0AE219DB5B

#	Filename of dependency	SHA1 digest	Scope
7	UserAgentUtils-1.18.jar	41982CF6B5B321B65FE3B45C7F1B59CB1512E306	COMPILE
8	DuoWeb-1.1.jar	24D0DDF6726D8F9CCA5AABBD07C4A60215AD6A66	COMPILE
9	idwsfconsumer-1.0.0.jar	99302B79C4D30BA0FF25B98E0353B10BFAC05F8D	COMPILE
10	javassist-3.18.1-GA.jar	D9A09F7732226AF26BF99F19E2CFFE0AE219DB5B	TEST

Table 3: Table with archives unknown to Maven Central

The following table contains 58 dependencies of the project under analysis (out of 147 distinct dependencies across all 30 project modules) with a release date older than 5 years ago.

Outdated dependencies with scopes COMPILE and RUNTIME are of primary interest, because they are expected and exposed at application runtime. Outdated dependencies with scope TEST have been omitted. For those with scope, it should be checked whether they exist as-is in the runtime environment, or whether a more recent version is provided therein.

#	Filename of dependency	Scope	Release Date
1	activation-1.1.jar	COMPILE	02/05/2006
2	ognl-2.6.11.jar	COMPILE	18/02/2007
3	commons-lang-2.4.jar	COMPILE	19/03/2008
4	stax-api-1.0-2.jar	COMPILE	04/10/2008
5	velocity-1.7.jar	COMPILE	29/11/2010
6	mail-1.4.7.jar	COMPILE	09/03/2013
7	javax.json-api-1.0.jar	COMPILE	24/04/2013
8	stax2-api-3.1.4.jar	COMPILE	28/02/2014
9	ant-1.9.4.jar	COMPILE	30/04/2014
10	ant-launcher-1.9.4.jar	COMPILE	30/04/2014
11	woodstox-core-asl-4.4.1.jar	COMPILE	12/09/2014
12	httpcore-4.3.3.jar	COMPILE	18/10/2014
13	httpclient-4.3.6.jar	COMPILE	02/11/2014
14	httpclient-cache-4.3.6.jar	COMPILE	02/11/2014
15	commons-codec-1.10.jar	COMPILE	06/11/2014
16	janino-2.7.8.jar	COMPILE	30/01/2015
17	commons-compiler-2.7.8.jar	COMPILE	30/01/2015
18	slf4j-api-1.7.12.jar	COMPILE	26/03/2015
19	metrics-core-3.1.2.jar	COMPILE	26/04/2015
20	metrics-json-3.1.2.jar	COMPILE	26/04/2015
21	xmlsec-2.0.5.jar	COMPILE	10/07/2015
22	jsr305-3.0.1.jar	COMPILE	09/10/2015
23	joda-time-2.9.jar	COMPILE	24/10/2015

#	Filename of dependency	Scope	Release Date
24	jai-imageio-core-1.3.1.jar	COMPILE	09/11/2015
25	commons-collections-3.2.2.jar	COMPILE	12/11/2015
26	guava-19.0.jar	COMPILE	09/12/2015
27	bcprov-jdk15on-1.54.jar	COMPILE	30/12/2015
28	opensaml-xmlsec-impl-3.2.0.jar	COMPILE	27/04/2016
29	okio-1.8.0.jar	COMPILE	02/05/2016
30	okhttp-3.3.0.jar	COMPILE	25/05/2016
31	retrofit-2.1.0.jar	COMPILE	15/06/2016
32	spring-webflow-2.4.4.RELEASE.jar	COMPILE	20/07/2016
33	spring-binding-2.4.4.RELEASE.jar	COMPILE	20/07/2016
34	spring-js-2.4.4.RELEASE.jar	COMPILE	20/07/2016
35	spring-js-resources-2.4.4.RELEASE.jar	COMPILE	20/07/2016
36	spring-context-support-4.3.2.RELEASE.jar	COMPILE	28/07/2016
37	spring-aop-4.3.2.RELEASE.jar	COMPILE	28/07/2016
38	spring-beans-4.3.2.RELEASE.jar	COMPILE	28/07/2016
39	spring-context-4.3.2.RELEASE.jar	COMPILE	28/07/2016
40	spring-core-4.3.2.RELEASE.jar	COMPILE	28/07/2016
41	spring-expression-4.3.2.RELEASE.jar	COMPILE	28/07/2016
42	spring-web-4.3.2.RELEASE.jar	COMPILE	28/07/2016
43	spring-webmvc-4.3.2.RELEASE.jar	COMPILE	28/07/2016
44	json-20160810.jar	COMPILE	10/08/2016
45	cryptacular-1.1.1.jar	COMPILE	10/08/2016
46	core-3.3.0.jar	COMPILE	16/09/2016
47	javase-3.3.0.jar	COMPILE	16/09/2016
48	jackson-annotations-2.8.3.jar	COMPILE	18/09/2016
49	jackson-core-2.8.3.jar	COMPILE	18/09/2016
50	jackson-databind-2.8.3.jar	COMPILE	18/09/2016
51	jackson-datatype-joda-2.8.3.jar	COMPILE	18/09/2016
52	jsonapi-converter-0.5.jar	COMPILE	21/10/2016
53	ldaptive-1.0.9.jar	COMPILE	02/11/2016
54	jsp-api-2.1.jar	PROVIDED	17/07/2006
55	jstl-1.2.jar	PROVIDED	23/06/2011
56	javax.servlet-api-3.0.1.jar	PROVIDED	12/07/2011
57	logback-core-1.1.3.jar	RUNTIME	24/03/2015
58	bcpkix-jdk15on-1.54.jar	RUNTIME	30/12/2015

Table 4: Archives with release date older than 5 years

3.2.6.1 Deviations from test procedure

None

3.3 *T1195.001-S3_TC1*

3.3.1 Security Requirements addressed

The use of open-source dependencies does not only come with the risk of vulnerabilities that have been accidentally created by benign open-source developers, but also with the risk of supply chain attacks, where rogue open-source contributors deliberately hide vulnerable or malicious code in open-source components. In this context, the scope of dependencies does not matter any more.

To reduce the attack surface, it is therefore worthwhile to remove all dependencies that are not actually needed for the project under analysis. The presence of un-used or bloated dependencies can be due to different reasons, e.g. when developers forget to remove old components, or if the need for transitive dependencies is not well understood.

The corresponding requirements are as follows:

- The project must be regularly checked regarding the presence of un-used bloated dependencies. This can be achieved using the static and dynamic analysis techniques of Eclipse Steady.
- In case neither static nor dynamic analysis find any constructs of a given dependency reachable, project developer shall check whether the dependency can be removed altogether.

3.3.2 Test preconditions

- A build job ran Steady's APP, A2C, PREPARE-AGENT, TEST and UPLOAD analysis goals on the project under analysis, in order to collect as much information as possible about the reachability of dependency code.
- The Steady <workspace> used for persisting the analysis results is known to the evaluator.

3.3.3 Expected test results

The test will result in a table of open-source components that were found to be unreachable, i.e. neither the static nor the dynamic analysis showed that any of its classes is needed.

Each of the dependencies shall be checked by application developers to see whether it can be removed altogether, herewith minimizing the attack surface and future maintenance efforts.

3.3.4 Criteria for evaluating results

Number of reachable constructs per component, as visible on each module's dependency tab, plus the assessment by developers for components that were not found to be reachable.

3.3.5 Test Procedure

- Point your browser to the Steady Web frontend (http://<host>:8033/apps/#/<workspace>) → Select all modules of the project under analysis (using the table on the left-hand side) → For each module: open the tab "Dependencies" on the right-hand side (cf. example screenshot in Figure 4) → Document all dependencies with 0 in column "Static analysis" and 0 in column "Dynamic analysis" → Present the findings to the developer(s)/architect(s) of the project to get an assessment for each component whether it is required or whether it can be removed

3.3.6 Test Results

Status: PASSED WITH DEVIATIONS

The following table shows 69 components of the project under analysis (out of 147 distinct dependencies across all 30 project modules) that have not been found reachable.

Again, the 14 unused components with scope COMPILE and RUNTIME are of primary interest, because they can contain deserialization gadgets that can be exploited at application runtime. However, in order to minimize the risk of supply chain attacks, it is advisable to also check whether any of the other dependencies can be removed.

Filename of dependency	Scope
activation-1.1.jar	COMPILE
jackson-annotations-2.8.3.jar	COMPILE
mail-1.4.7.jar	COMPILE
ognl-2.6.11.jar	COMPILE
opensaml-saml-api-3.4.0.jar	COMPILE
opensaml-soap-api-3.4.0.jar	COMPILE
spring-js-resources-2.4.4.RELEASE.jar	COMPILE
stax-api-1.0-2.jar	COMPILE
stax2-api-3.1.4.jar	COMPILE
jai-imageio-core-1.3.1.jar	COMPILE
okhttp-3.3.0.jar	COMPILE
okio-1.8.0.jar	COMPILE
retrofit-2.1.0.jar	COMPILE
jstl-1.2.jar	PROVIDED
bcpkix-jdk15on-1.54.jar	RUNTIME
ant-1.7.0.jar	TEST
ant-launcher-1.7.0.jar	TEST
bsh-2.0b4.jar	TEST
hamcrest-core-1.1.jar	TEST
java-support-7.3.0-tests.jar	TEST
jcl-over-slf4j-1.7.12.jar	TEST
jul-to-slf4j-1.7.12.jar	TEST
junit-4.10.jar	TEST
log4j-over-slf4j-1.7.12.jar	TEST
snakeyaml-1.15.jar	TEST
testng-6.9.9.jar	TEST
xmlunit-1.6.jar	TEST
antlr-2.7.7.jar	TEST
dom4j-1.6.1.jar	TEST
hibernate-commons-annotations-4.0.4.Final.jar	TEST
hibernate-core-4.3.5.Final.jar	TEST
hibernate-entitymanager-4.3.5.Final.jar	TEST

Filename of dependency	Scope
hibernate-jpa-2.1-api-1.0.0.Final.jar	TEST
jandex-1.1.0.Final.jar	TEST
javassist-3.18.1-GA.jar	TEST
jboss-logging-3.1.3.GA.jar	TEST
jboss-logging-annotations-1.2.0.Beta1.jar	TEST
jboss-transaction-api_1.2_spec-1.0.0.Final.jar	TEST
jetty-http-9.2.14.v20151106.jar	TEST
jetty-io-9.2.14.v20151106.jar	TEST
jetty-server-9.2.14.v20151106.jar	TEST
jetty-util-9.2.14.v20151106.jar	TEST
mockito-core-1.10.8.jar	TEST
objenesis-2.1.jar	TEST
spring-jdbc-4.3.2.RELEASE.jar	TEST
spring-orm-4.3.2.RELEASE.jar	TEST
spring-tx-4.3.2.RELEASE.jar	TEST
spymemcached-2.11.4.jar	TEST
xml-apis-1.0.b2.jar	TEST
opensaml-profile-api-3.3.0-tests.jar	TEST
opensaml-saml-api-3.3.0-tests.jar	TEST
antlr-runtime-3.4.jar	TEST
jna-4.1.0.jar	TEST
jna-platform-4.1.0.jar	TEST
jsch.agentproxy.connector-factory-0.0.7.jar	TEST
jsch.agentproxy.core-0.0.7.jar	TEST
jsch.agentproxy.pageant-0.0.7.jar	TEST
jsch.agentproxy.sshagent-0.0.7.jar	TEST
jsch.agentproxy.svnkit-trilead-ssh2-0.0.7.jar	TEST
jsch.agentproxy.usocket-jna-0.0.7.jar	TEST
jsch.agentproxy.usocket-nc-0.0.7.jar	TEST
platform-3.4.0.jar	TEST
sequence-library-1.0.3.jar	TEST
spring-extensions-5.3.0-tests.jar	TEST
trilead-ssh2-1.0.0-build220.jar	TEST
hsqldb-2.3.3.jar	TEST
unboundid-ldapsdk-2.3.8.jar	TEST

Filename of dependency	Scope
mockito-all-1.10.8.jar	TEST
opensaml-soap-impl-3.3.0-tests.jar	TEST

Table 5: Potentially un-used archives with scope compile, runtime and provided

3.3.6.1 Deviations from test procedure

The analysis by the developers/architects is outstanding.

3.3.6.2 Problems encountered

None

Chapter 4 Test Summary Coverage

This chapter shows the completeness of tests coverage: each test covers at least one requirement, and every requirement has been tested at least by one test.

The following Table 6 demonstrates that each test cover at least one requirement.

Test ID	Requirement code	Results (including section reference)	Notes
T1195.001-S1_TC1	T1195.001-S1	FAILED (3.1.6)	The project depends on open-source components with known vulnerabilities. They must be assessed and either fixed or exempted (providing a justification why the vulnerability is not critical/exploitable).
T1195.001-S2_TC1	T1195.001-S2	FAILED (3.2.6)	The project depends on 58 outdated components with scope COMPILE and RUNTIME (released >5 years ago) and on 10 components unknown to Maven Central.
T1195.001-S3_TC1	T1195.001-S3	PASSED WITH DEVIATIONS (3.3.6)	The project depends on 69 components that were not found to be reachable, the developer assessment is outstanding.

Table 6: Test Summary Coverage (Tests vs Requirements)

The following Table 6 demonstrates that each requirement has been verified at least through one test.

Requirement code	Test ID	Results (including section reference)	Notes
T1195.001-S1	T1195.001-S1_TC1	FAILED (3.1.6)	The project depends on open-source components with known vulnerabilities. They must be assessed and either fixed or exempted (providing a justification why the vulnerability is not critical/exploitable).
T1195.001-S2	T1195.001-S2_TC1	FAILED (3.2.6)	The project depends on 58 outdated components with scope COMPILE and RUNTIME (released >5 years ago) and on 10 components unknown to Maven Central.
T1195.001-S3	T1195.001-S3_TC1	PASSED WITH DEVIATIONS (3.3.6)	The project depends on 69 components that were not found to be reachable, the developer assessment is outstanding.

Table 7: Test Summary Coverage (Requirements vs Test)

The following matrix (Table 8) shows the complete coverage between Security Functional Requirements and tests

	T1195.001-S1	T1195.001-S2	T1195.001-S3
T1195.001-S1_TC1	X		
T1195.001-S2_TC1		X	
T1195.001-S3_TC1			X

Table 8: Matrix of test coverage

Chapter 5 List of Abbreviations

Abbreviation	Translation
API	Application Programming Interface
App	Application
ATE	Assurance Family Test
CAPE	Continuous Assessment in Polymorphous Environments
CI/CD	Continuous Integration/Continuous Delivery
CIE	Carta d'Identità Elettronica
CINI	Consorzio Interuniversitario Nazionale per l'Informatica
CVE	Common Vulnerabilities and Exposures
CVSS	Common Vulnerability Scoring System
DevSecOps	Development Security Operations
FBK	Fondazione Bruno Kessler
GB	Giga Byte
GitLab	Open source end-to-end software development platform
GitLab-CI/CD	GitLab-Continuous Integration/Continuous Delivery
HTML	HyperText Markup Language
HTTP/HTTPS	Hypertext Transfer Protocol / Hypertext Transfer Protocol Secure
OS	Operating System
OWASP	Open Web Application Security Project
RAM	Random Access Memory
REST	Representational State Transfer
SAML	Security Assertion Markup Language
SAML IdP	Security Assertion Markup Language Identity Provider
SHA-*	Secure Hash Algorithm-*
SRs	Security Requirements
SSO	Single Sign-On
VM	Virtual Machine

Chapter 6 Bibliography

[1] SPARTA CAPE D5.2 “Demonstrator specifications”, January 2021

[2] SPARTA CAPE D5.3 “Demonstrator prototypes”, January 2021

[3] SPARTA CAPE D5.4 “Integration on demonstration cases and validation”, January 2022

[4] Common Criteria for Information Technology Security Evaluation, Version 3.1, revision 5, April 2017. Part 1: Introduction and general model.

[5] Common Criteria for Information Technology Security Evaluation, Version 3.1, revision 5, April 2017. Part 3: Assurance security components.

[6] Bundesamt für Sicherheit in der Informationstechnik (BSI) Guidelines for Developer Documentation according to Common Criteria Version 3.1 Version 1.0