
Processo di Validazione

Release 3.3.12

Link.it

17 apr 2023

1	Introduzione	1
2	Static Code Analysis	3
2.1	Sorgenti soggetti a controllo qualità	4
2.2	SpotBugs	7
2.3	SonarQube	20
3	Dynamic Analysis	27
3.1	Security Tests	27
3.2	Functional tests	35
4	Third Party Dependency Analysis	51
4.1	OWASP Dependency-Check Jenkins Plugin	52
4.2	OWASP Dependency-Check Maven Plugin	55
4.3	OWASP Dependency-Check Falsi Positivi	55

Ogni nuova versione di GovWay prima del rilascio viene sottoposta a 3 diversi tipi di verifiche di sicurezza al fine di assicurarne la stabilità e l'assenza di vulnerabilità note.

- *Static Code Analysis*: identifica possibili vulnerabilità all'interno del codice sorgente tramite i tools [SpotBugs](#) e [SonarQube](#).
- *Dynamic Analysis*: cerca vulnerabilità del software durante l'effettiva esecuzione del prodotto. L'analisi viene eseguita attraverso l'esecuzione di estese batterie di test realizzate tramite i tool [TestNG](#), [JUnit](#), [Karate](#) e [OWASP ZAP Proxy](#).
- *Third Party Dependency Analysis*: assicura che tutte le librerie terza parte utilizzate non siano soggette a vulnerabilità di sicurezza note, utilizzando il tool [OWASP Dependency-Check](#).

Ognuna di tali fasi viene anche verificata per ogni commit sul [master](#) dei sorgenti del progetto nell'ambiente di [Continuous Integration Jenkins](#) di GovWay.

Static Code Analysis

In questa fase vengono identificate possibili vulnerabilità all'interno del codice sorgente cercando pattern riconducibili a bug improbabili da individuare tramite test dinamici (*Dynamic Analysis*).

L'analisi viene effettuata tramite l'utilizzo dei seguenti tool:

- [SpotBugs](#)
- [SonarQube](#)

Il tool *SpotBugs* viene utilizzato fin dalle fasi di sviluppo dai programmatori tramite il [plugin per Eclipse](#) come descritto nella sezione *SpotBugs Eclipse Plugin*.

Dopo aver lavorato su un branch dedicato alla realizzazione di una nuova funzionalità o di un bug fix, prima di riportare il lavoro sul master i sorgenti soggetti a modifica vengono verificati anche tramite il tool *SonarQube* (*SonarLint Eclipse Plugin*).

Ad ogni commit sul [master dei sorgenti del progetto](#) viene effettuata automaticamente una verifica dei sorgenti nell'ambiente di [Continuous Integration Jenkins di GovWay](#) utilizzando entrambi i tools. Maggiori dettagli vengono forniti nelle sezioni *SpotBugs Warnings Jenkins Plugin* e *SonarQube Jenkins Plugin*.

Infine effettuato il checkout dei [sorgenti del progetto GovWay](#), è sempre possibile avviare manualmente una analisi statica tramite uno dei tool seguendo le indicazioni fornite nelle sezioni *SpotBugs Maven Plugin* e *SonarQube Maven Plugin*.

Nota: I problemi di sicurezza relativi alle librerie terza parte utilizzate sono trattati separatamente con il lavoro di analisi descritto nella sezione *Third Party Dependency Analysis* e di conseguenza il codice sorgente di tali librerie è escluso dall'analisi del codice statico.

2.1 Sorgenti soggetti a controllo qualità

Nei *sorgenti del progetto GovWay* sono presenti sia i moduli utilizzati dagli archivi binari di GovWay che i componenti che realizzano i test dinamici (*Dynamic Analysis*); quest'ultimi non sono soggetti a controllo qualità. Di seguito viene riportato per ogni modulo soggetto a controllo qualità la posizione all'interno del progetto.

- *utilità di base*; libreria di utility comune utilizzata da tutti gli altri componenti:
 - *tools/utls* (archivio: *openspcoop2_utls-<version>.jar*)
 - *tools/generic_project* (archivio: *openspcoop2_generic-project-<version>.jar*)
- *runtime gateway*; contiene i moduli che definiscono il runtime di govway:
 - *core/src/org/openspcoop2/message* (archivio: *openspcoop2_message-<version>.jar*)
 - *core/src/org/openspcoop2/core* (archivio: *openspcoop2_core-<version>.jar*)
 - *core/src/org/openspcoop2/protocol* (archivio: *openspcoop2_protocol-api-<version>.jar* e *openspcoop2_protocol-<version>.jar*)
 - *core/src/org/openspcoop2/monitor* (archivio: *openspcoop2_monitor-api-<version>.jar* e *openspcoop2_monitor-<version>.jar*)
 - *core/src/org/openspcoop2/security* (archivio: *openspcoop2_security-<version>.jar*)
 - *core/src/org/openspcoop2/pdd* (archivio: *openspcoop2_pdd-<version>.jar*)
- *profili di interoperabilità*; ogni profilo viene realizzato come un plugin che consente di personalizzare il comportamento del runtime:
 - “API Gateway”; *protocolli/trasparente* (archivio: *openspcoop2_trasparente-protocol-<version>.jar*)
 - “ModI”; *protocolli/modipa* (archivio: *openspcoop2_modipa-protocol-<version>.jar*)
 - “SPCoop”; *protocolli/spcoop* (archivio: *openspcoop2_spcoop-protocol-<version>.jar*)
 - “eDelivery”; *protocolli/as4* (archivio: *openspcoop2_as4-protocol-<version>.jar*)
 - “Fatturazione Elettronica”; *protocolli/sdi* (archivio: *openspcoop2_sdi-protocol-<version>.jar*)
- *console web*; di seguito vengono descritti tutti i moduli che definiscono le console di gestione e di monitoraggio:
 - console di gestione “govwayConsole”:
 - * *tools/web_interfaces/lib/control_station* (archivio: *openspcoop2_web-govwayConsole-<version>.jar*)
 - console di monitoraggio “govwayMonitor”:
 - * *tools/web_interfaces/lib/monitor/src/src_core* (archivio: *openspcoop2_web-govwayMonitor-core-<version>.jar*)
 - * *tools/web_interfaces/lib/monitor/src/src_transazioni* (archivio: *openspcoop2_web-govwayMonitor-transazioni-<version>.jar*)
 - * *tools/web_interfaces/lib/monitor/src/src_statistiche* (archivio: *openspcoop2_web-govwayMonitor-statistiche-<version>.jar*)
 - * *tools/web_interfaces/lib/monitor/src/src_eventi* (archivio: *openspcoop2_web-govwayMonitor-eventi-<version>.jar*)
 - * *tools/web_interfaces/lib/monitor/src/src_allarmi* (archivio: *openspcoop2_web-govwayMonitor-allarmi-<version>.jar*)
 - * “Pagine JSF”; *tools/web_interfaces/monitor/deploy/pages*

- librerie comuni:
 - * “Audit”; `tools/web_interfaces/lib/audit` (archivio: `openspcoop2_web-lib-audit-<version>.jar`)
 - * “Utenze”; `tools/web_interfaces/lib/users` (archivio: `openspcoop2_web-lib-users-<version>.jar`)
 - * “Code”; `tools/web_interfaces/lib/queue` (archivio: `openspcoop2_web-lib-queue-<version>.jar`)
 - * “Widget”; `tools/web_interfaces/lib/mvc` (archivio: `openspcoop2_web-lib-mvc-<version>.jar`)
 - * “Loader”; `tools/web_interfaces/loader` (archivio: `openspcoop2_web-loaderConsole-<version>.jar`)
 - * “Javascript”; `tools/web_interfaces/lib/js`
 - * “Pagine JSP”; `tools/web_interfaces/lib/jsplib`
- *api*; le api di configurazione e monitoraggio:
 - api di configurazione “govwayAPIConfig”; `tools/rs/config/server` (archivio: `openspcoop2_rs-config-server-<version>.jar`)
 - api di configurazione “govwayAPIMonitor”; `tools/rs/monitor/server` (archivio: `openspcoop2_rs-monitor-server-<version>.jar`)
- *batch*; i batch utilizzati a run time in GovWay:
 - batch di generazione delle statistiche; `tools/batch/statistiche` (archivio: `openspcoop2_batch-statistiche-<version>.jar`)
 - batch per la gestione del repository di runtime; `tools/batch/runtime-repository` (archivio: `openspcoop2_batch-runtime-repository-<version>.jar`)

2.1.1 Eclipse Project

Nei *sorgenti del progetto GovWay* sia i moduli utilizzati dagli archivi binari di GovWay che i componenti che realizzano i test dinamici sono sviluppati tramite progetti eclipse. Di seguito vengono fornite le indicazioni su come importarli:

- nella radice del progetto utilizzare la fase maven “initialize” per salvare le librerie 3parti nella directory lib, tramite il comando “mvn initialize”;
- accedere alla sezione «Window -> Preferences -> Java -> Installed JREs» (Fig. 2.1) verificando che esista una jre v11 (aggiungerla se non esiste) e che sia selezionata come opzione di default (in alternativa si dovrà impostare l'utilizzo in ogni progetto importato nel seguito);
- accedere alla sezione «Window -> Preferences -> Java -> Build Path -> User Libraries», come mostrato nella figura Fig. 2.2;
- importare le librerie `lib/openspcoop2.userlibraries` e `lib/openspcoop2-testsuite.userlibraries` (Fig. 2.3);
- importate le 2 librerie, accedere alla sezione «File -> Import -> General - Existing Projects into Workspace» come mostrato nella figura Fig. 2.4;
- selezionando la directory contenente i *sorgenti del progetto GovWay* verrà proposto il progetto eclipse “op2_3.x.dev” (Fig. 2.5);
- durante l'import cliccando sulla voce “Search for nested projects” verranno importati anche tutti gli altri progetti che realizzano i test dinamici descritti in *Dynamic Analysis* (Fig. 2.6);
- una volta importati i progetti saranno disponibili come mostrato nella figura Fig. 2.7.

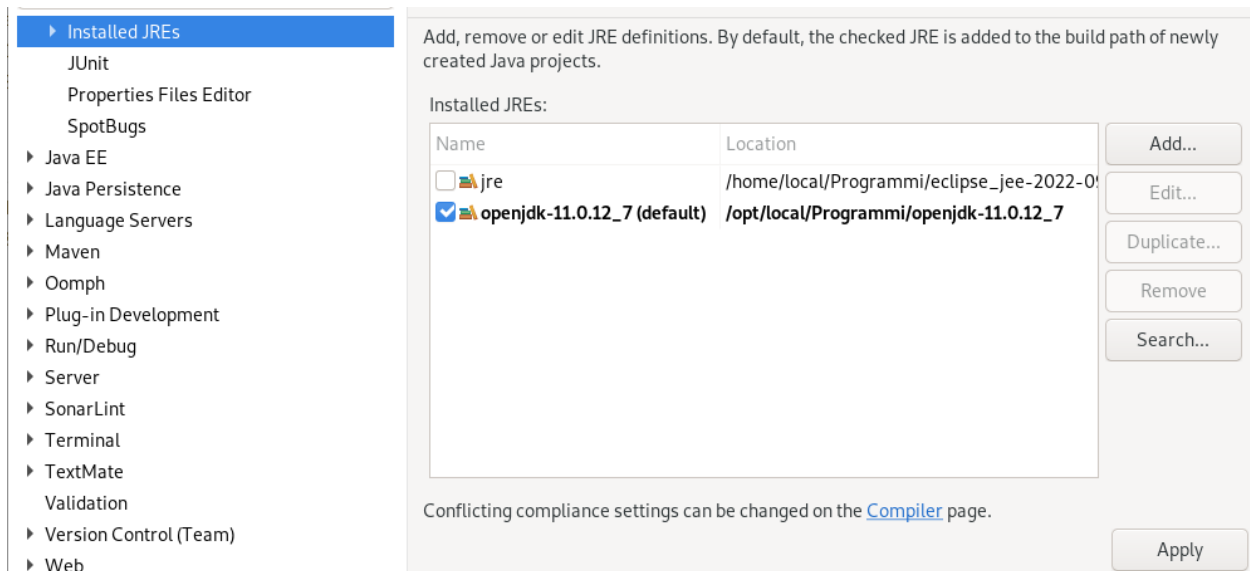


Fig. 2.1: Eclipse Project: jre libraries

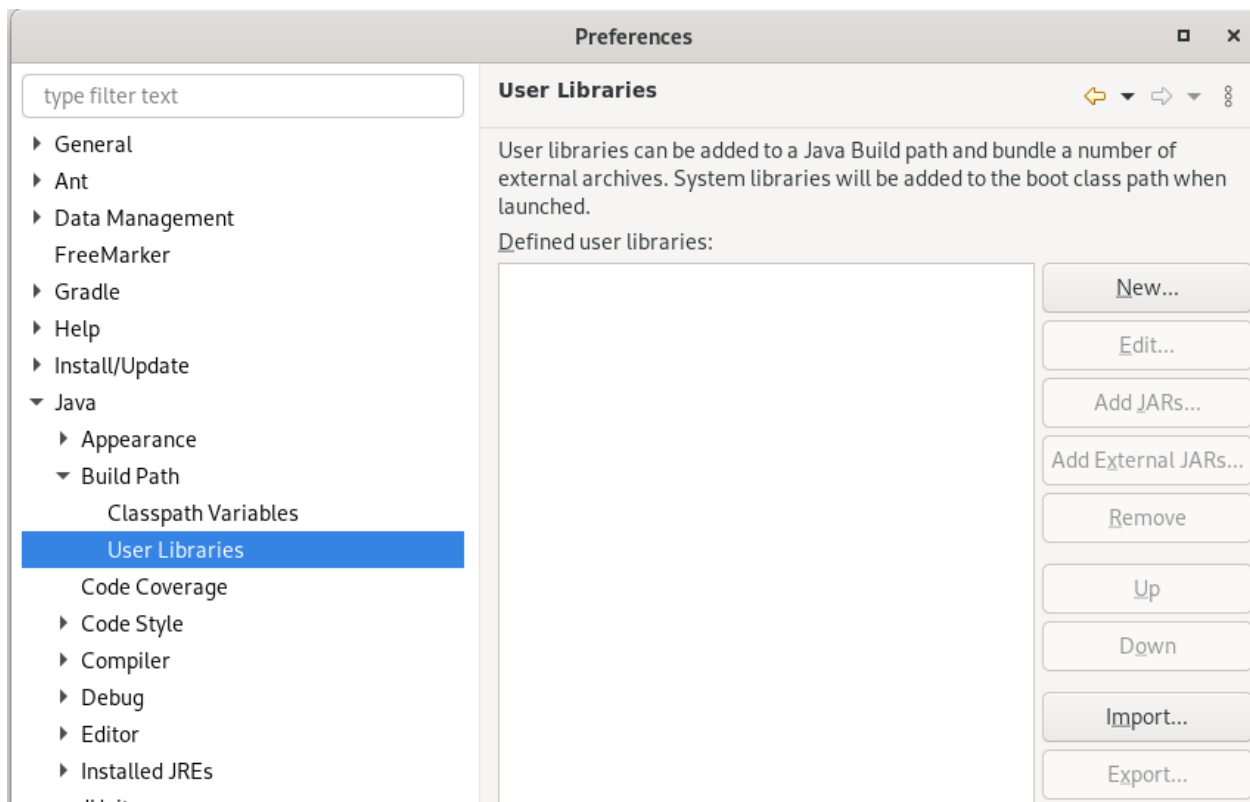


Fig. 2.2: Eclipse Project: import user libraries

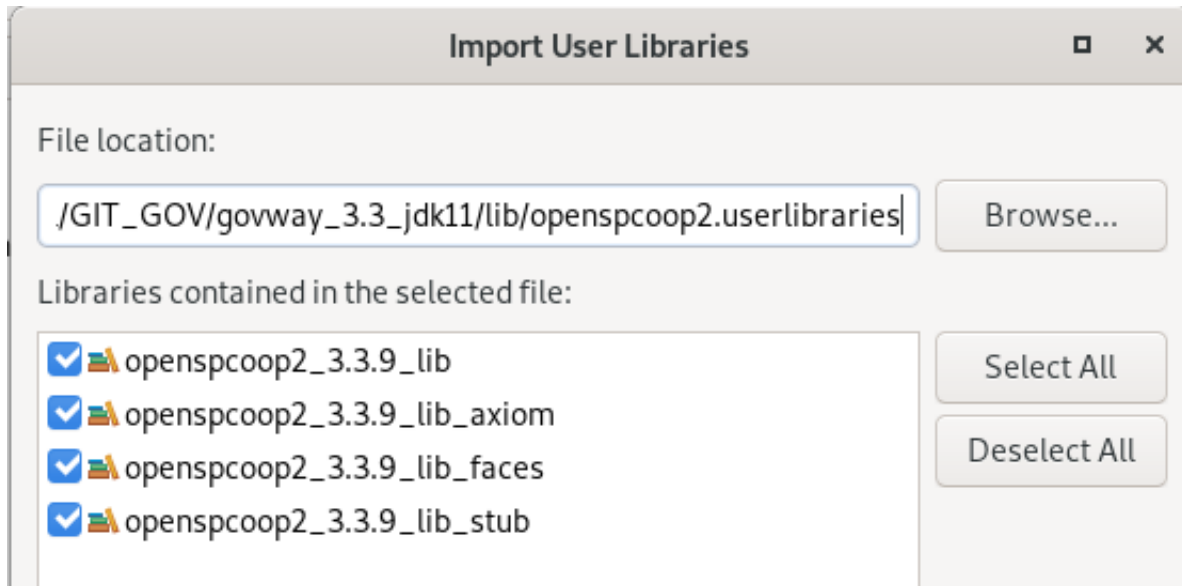


Fig. 2.3: Eclipse Project: import govway libraries

2.2 SpotBugs

In questa fase vengono identificate possibili vulnerabilità all'interno del codice sorgente tramite il tool [SpotBugs](#).

Il tool viene utilizzato fin dalle fasi di sviluppo dai programmatori tramite il [plugin per Eclipse](#) come descritto nella sezione [SpotBugs Eclipse Plugin](#).

Ad ogni commit sul [master dei sorgenti del progetto](#) viene effettuata automaticamente una verifica dei sorgenti nell'ambiente di [Continuous Integration Jenkins di GovWay](#). Maggiori dettagli vengono forniti nella sezione [SpotBugs Warnings Jenkins Plugin](#).

Una verifica manuale dei [sorgenti del progetto GovWay](#) è attuabile seguendo le indicazioni presenti nella sezione [SpotBugs Maven Plugin](#).

Configurazione

Di seguito vengono forniti i criteri di esecuzione dell'analisi statica tramite il tool "SpotBugs":

- **efforts**: viene utilizzato il livello "max" che consente di avere la massima precisione per trovare più bug (vedi sezione [efforts](#));
- **confidence**: viene utilizzato il livello "low" per non filtrare alcun bug (vedi parametro [confidence](#));
- **rank**: vengono verificati tutti i livelli di bug da 1 a 4 (quelli considerati "spaventosi"), quelli da 5 a 9 (gravi), da 10 a 14 (preoccupanti) e il livello 15 dei bug di basso profilo.

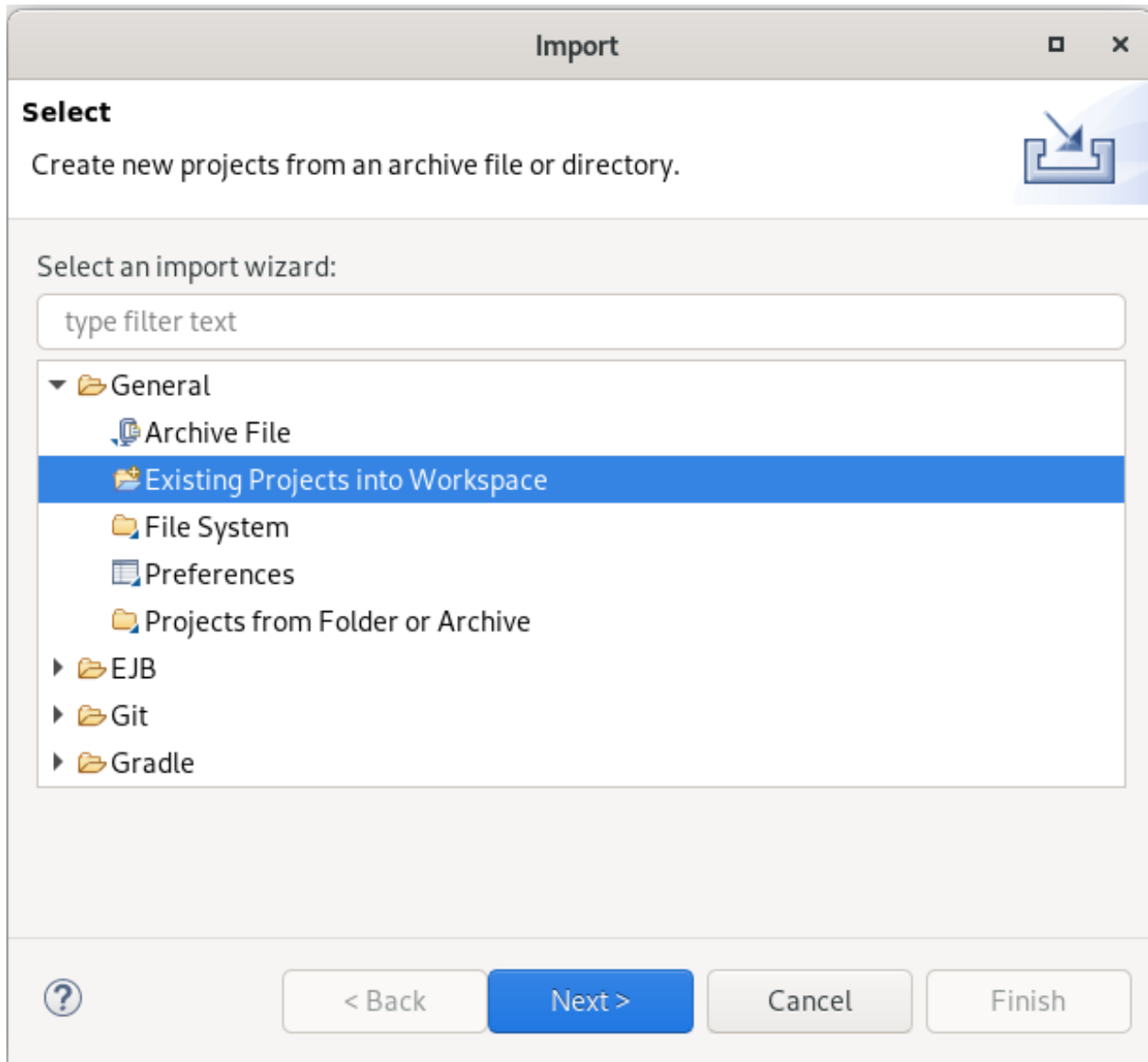


Fig. 2.4: Eclipse Project: import existing project

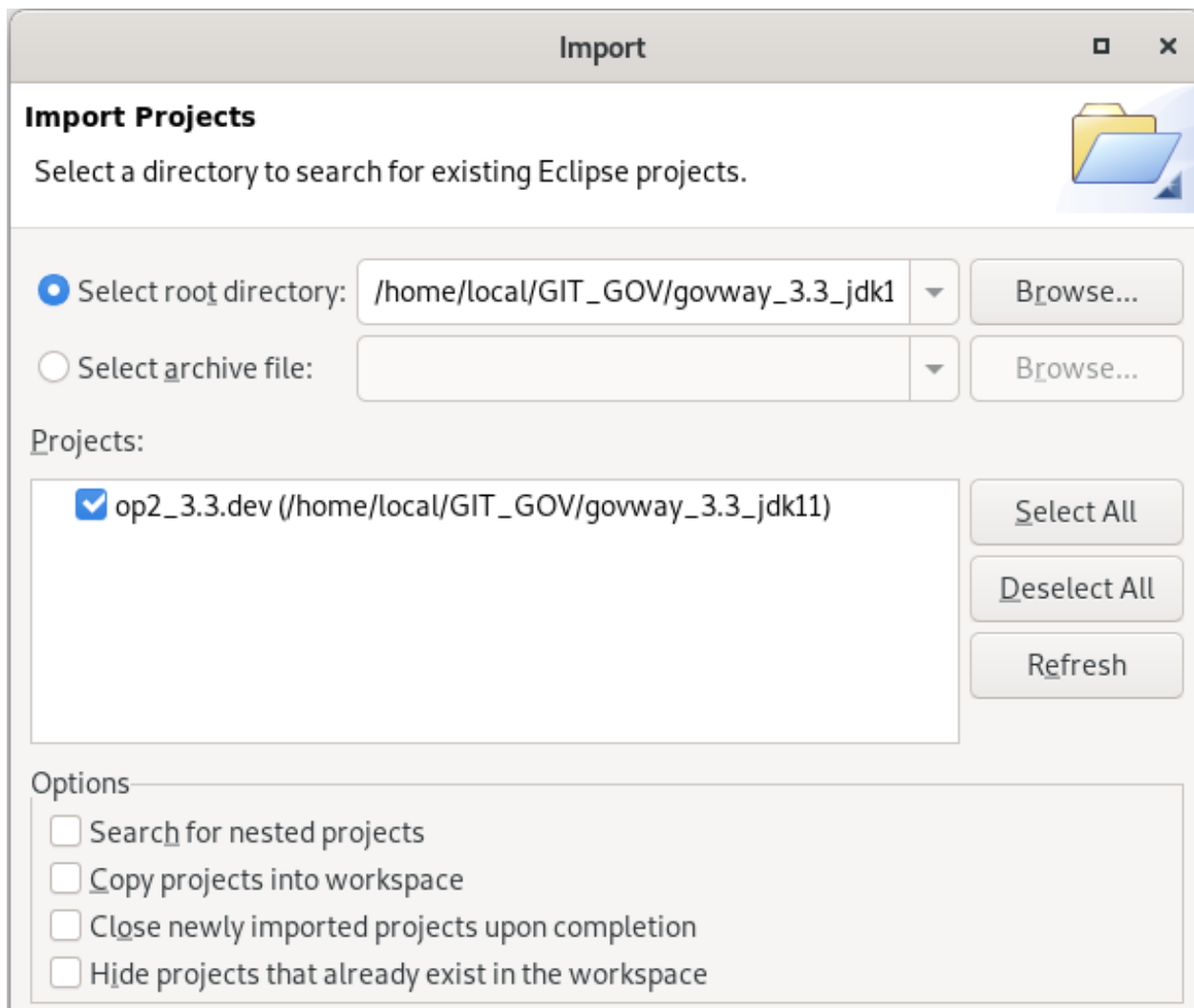


Fig. 2.5: Eclipse Project: import project op2_3.x.dev

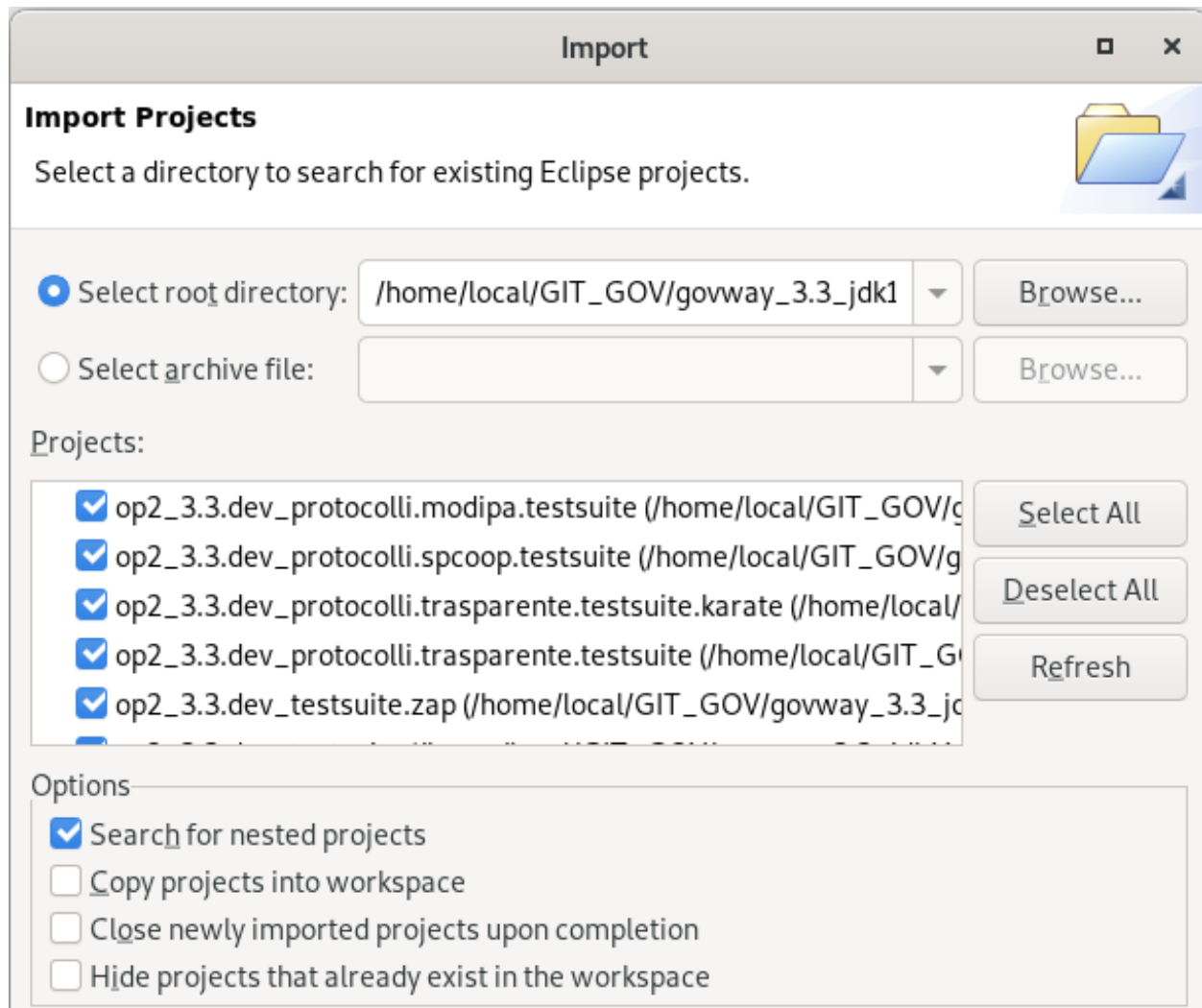


Fig. 2.6: Eclipse Project: import all project



```

▶ > op2_3.3.dev [govway_3.3_jdk11 master]
▶ > op2_3.3.dev_protocolli.modipa.testsuite [govway_3.3_jdk11 master]
▶ > op2_3.3.dev_protocolli.spcoop.testsuite [govway_3.3_jdk11 master]
▶ > op2_3.3.dev_protocolli.trasparente.testsuite [govway_3.3_jdk11 master]
▶ > op2_3.3.dev_protocolli.trasparente.testsuite.karate [govway_3.3_jdk11 master]
▶ > op2_3.3.dev_testsuite [govway_3.3_jdk11 master]
▶ > op2_3.3.dev_testsuite.zap [govway_3.3_jdk11 master]
▶ > op2_3.3.dev_tools.rs.config.server.testsuite [govway_3.3_jdk11 master]
▶ > op2_3.3.dev_tools.rs.monitor.server.testsuite [govway_3.3_jdk11 master]
▶ > op2_3.3.dev_tools.web_generic_project.impl.jsf1 [govway_3.3_jdk11 master]

```

Fig. 2.7: Eclipse Project: govway projects

2.2.1 SpotBugs Warnings Jenkins Plugin

Ad ogni commit sul [master dei sorgenti del progetto](#) viene effettuata automaticamente una verifica dei sorgenti nell'ambiente di [Continuous Integration Jenkins di GovWay](#).

L'analisi produce un [report di dettaglio](#) sulle vulnerabilità trovate. Per ogni vulnerabilità identificata vengono forniti maggiori dettagli come la severità, la categoria (es. Security), il tipo (codice del pattern che identifica il bug), il package e la classe dove è stato rilevato (es. [Fig. 2.8](#)).

Nella [homepage dell'ambiente CI Jenkins di GovWay](#) è anche disponibile un report che visualizza il trend delle vulnerabilità rispetto ai commit effettuati nel tempo (es. [Fig. 2.9](#)).

Sono inoltre disponibili [report di dettaglio in vari formati](#) ([Fig. 2.10](#)).

La figura [Fig. 2.11](#) mostra un esempio di report nel formato HTML.

2.2.2 SpotBugs Eclipse Plugin

In questa sezione viene descritto come utilizzare il [plugin per Eclipse](#) per la verifica del codice sorgente.

Come prerequisito il plugin deve essere stato installato tramite "Eclipse Marketplace" come mostrato nella figura [Fig. 2.12](#).

Impostare i seguenti criteri di analisi statica accedendo alla sezione «Window -> Preferences -> Java -> SpotBugs», come mostrato nella figura [Fig. 2.13](#):

- un livello "Maximal" per il parametro "Analysis Effort";
- un livello "Low" per il parametro "Minimum confidence to report";
- un "rank" impostato al valore "15".

Deve inoltre essere caricato il filtro che esclude alcuni [falsi positivi](#) come mostrato nella figura [Fig. 2.14](#).

L'analisi statica dei sorgenti è adesso effettuabile selezionando il progetto "op2_3.x.dev" (*Eclipse Project*) con il tasto destro e cliccando sulla voce "SpotBugs -> Find Bugs" come mostrato nella figura [Fig. 2.15](#).

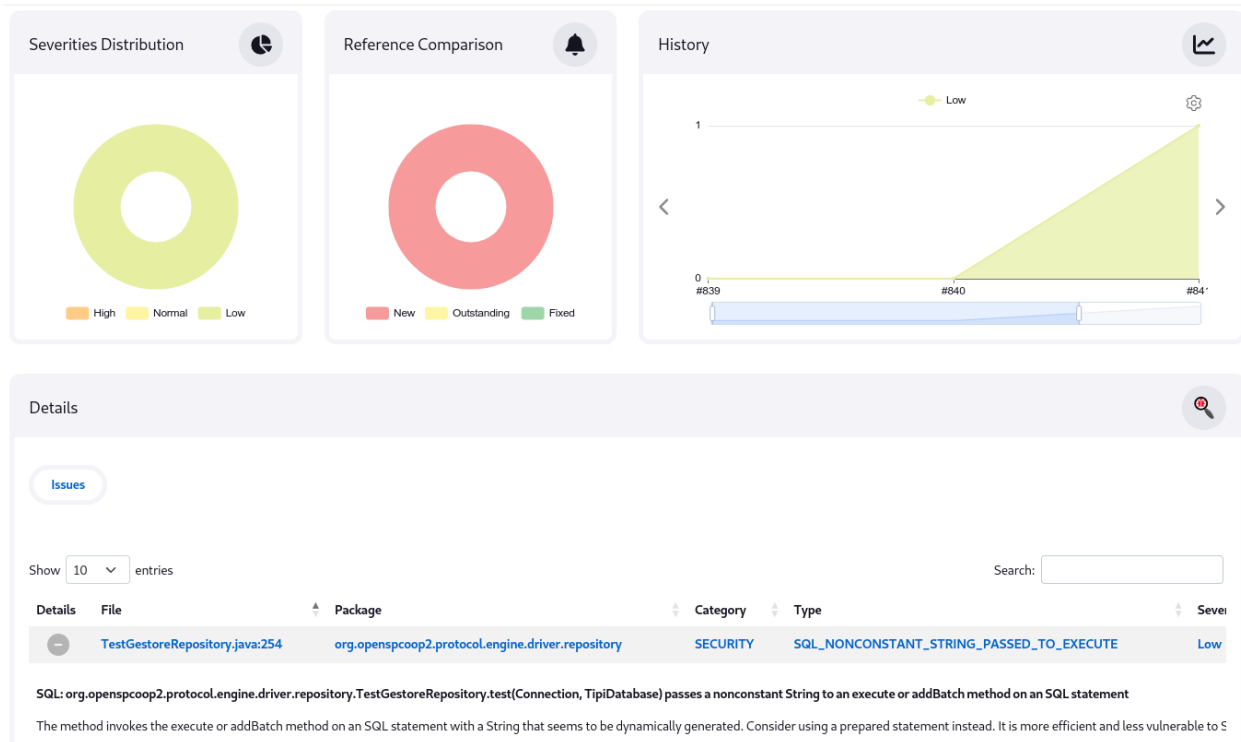


Fig. 2.8: SpotBugs: dettaglio di una vulnerabilità

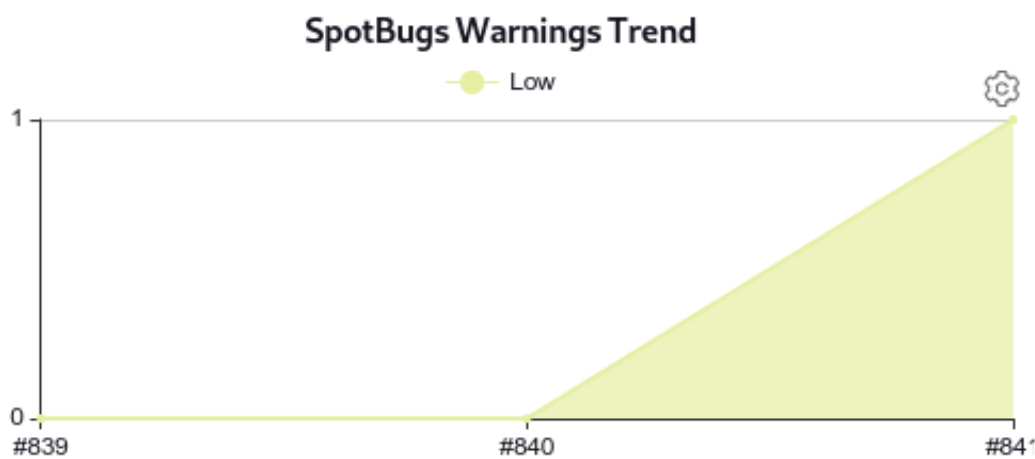


Fig. 2.9: SpotBugs Warnings Trend

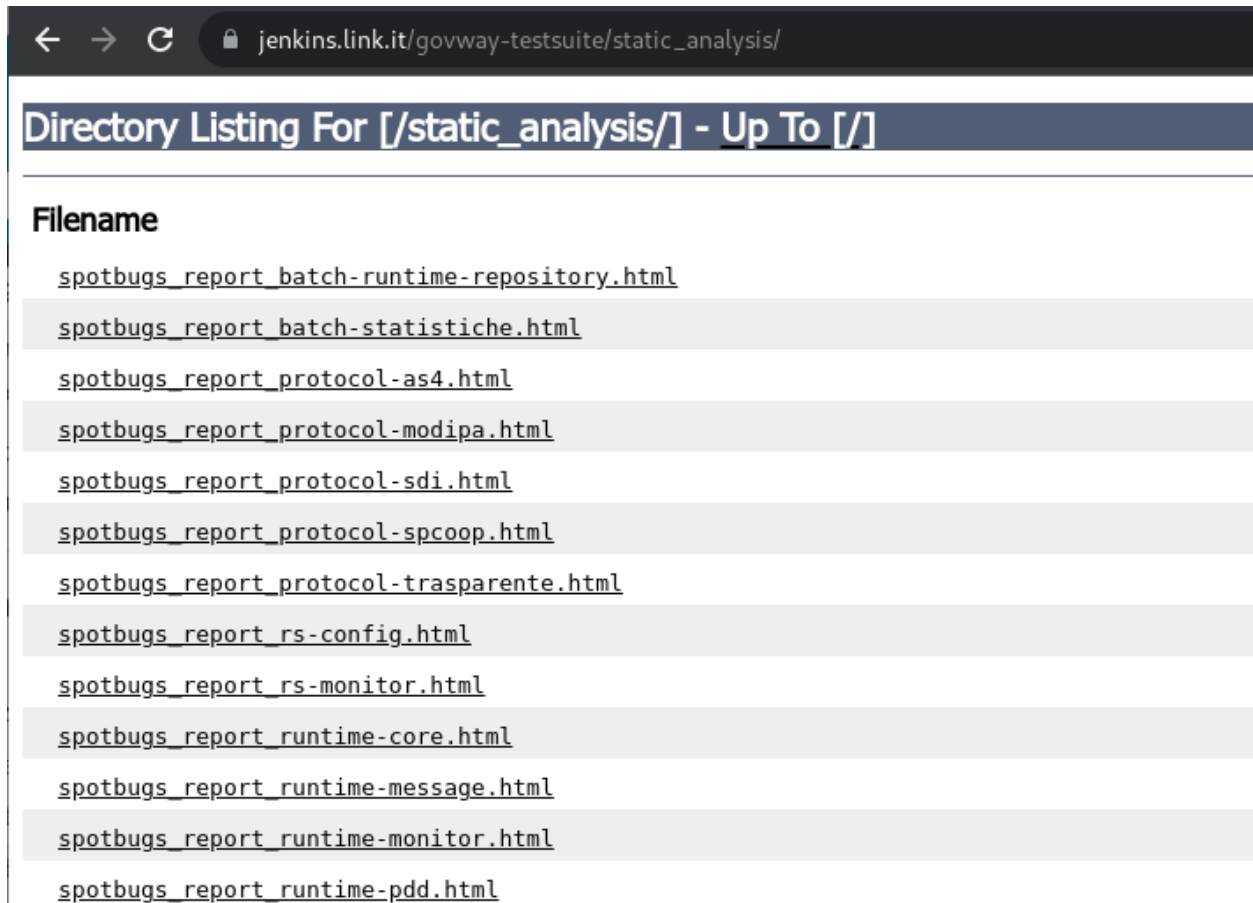


Fig. 2.10: SpotBugs: report in vari formati

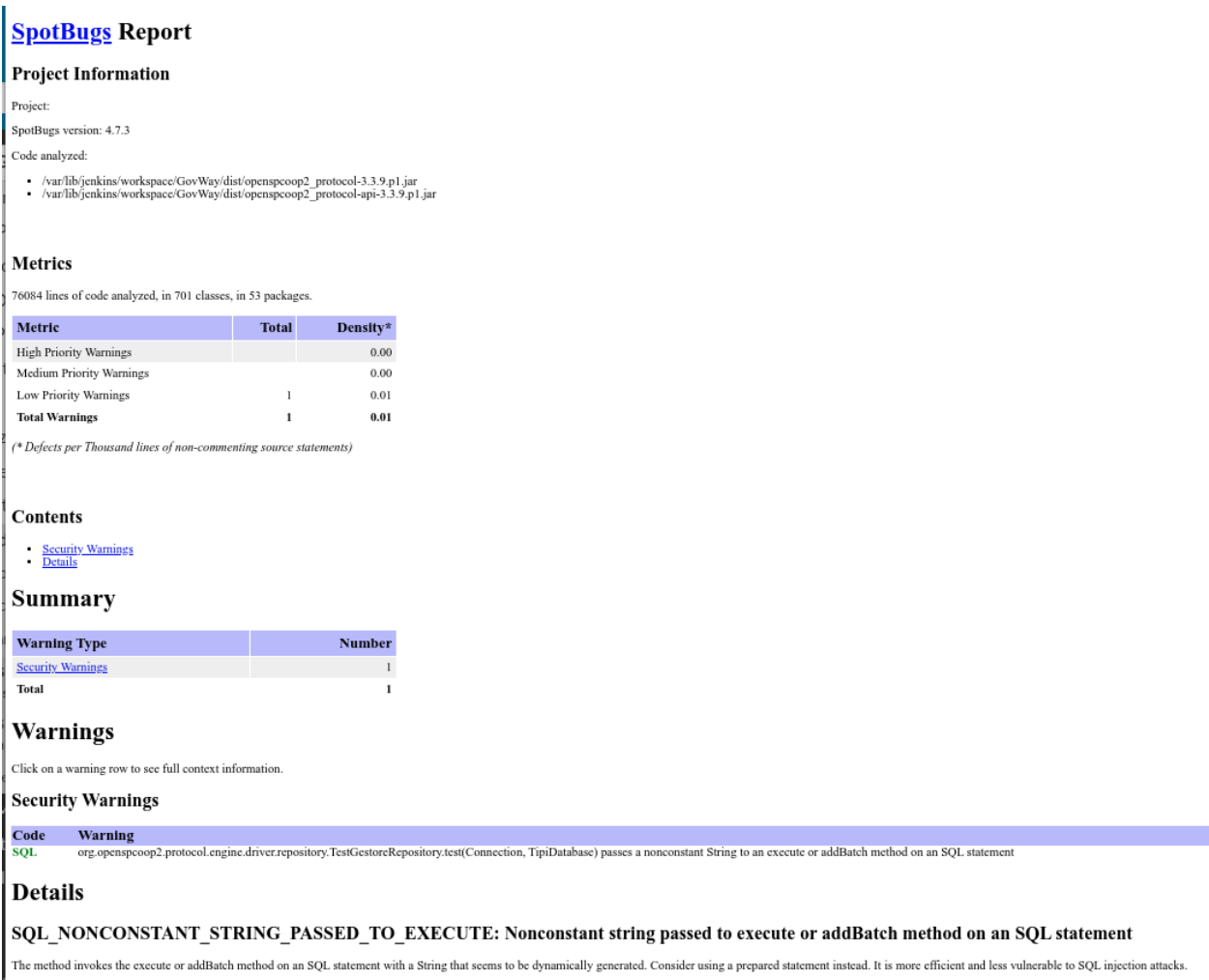


Fig. 2.11: SpotBugs: html report

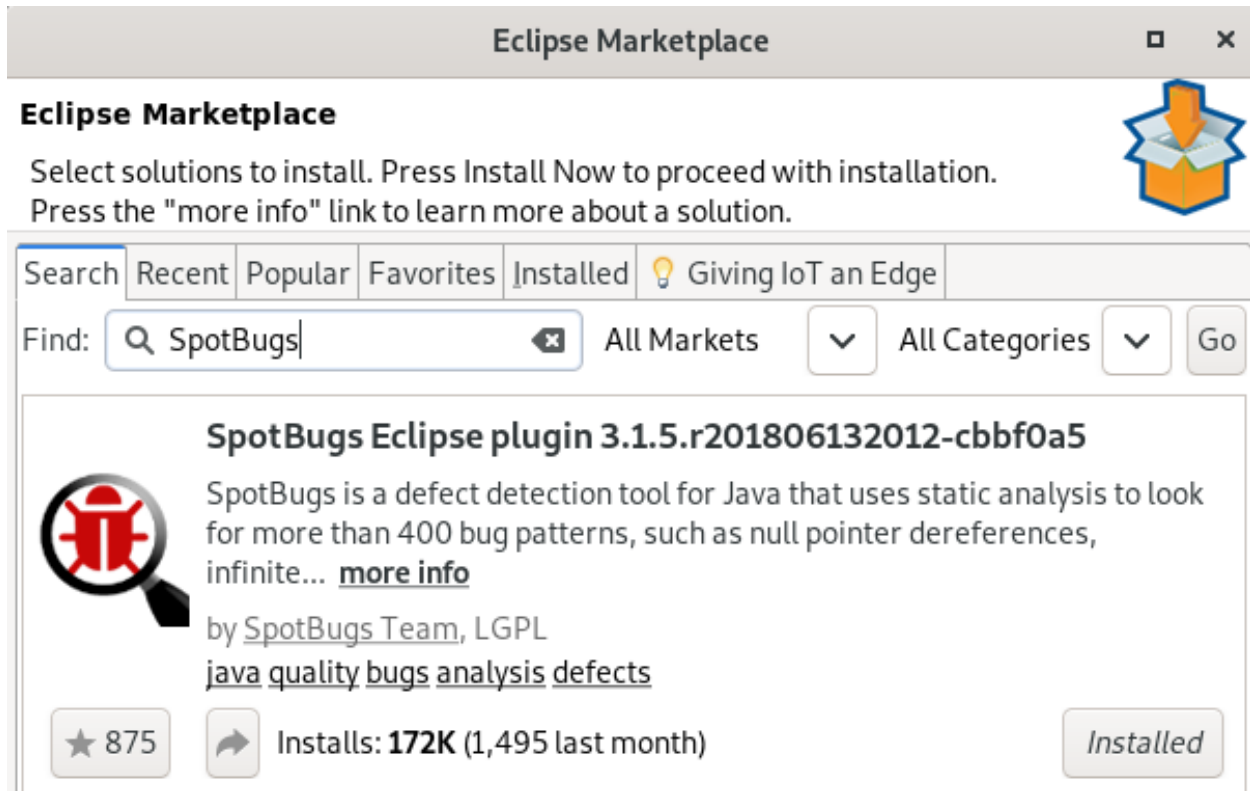


Fig. 2.12: SpotBugs Eclipse Plugin: marketplace

Eventuali bug individuati vengono evidenziati sulla singola classe come ad esempio viene mostrato nella figura Fig. 2.16.

2.2.3 SpotBugs Maven Plugin

Effettuato il checkout dei sorgenti del progetto GovWay, è possibile avviare manualmente l'analisi statica del codice utilizzando il seguente comando maven nella radice del progetto:

```
mvn verify -Dspotbugs=verify -Dspotbugs.home=PATH_ASSOLUTO_TOOLS_SPOTBUGS -
-Dtestsuite=none -Dpackage=none -Dowasp=none
```

Come prerequisito all'esecuzione deve essere effettuato il download dell'ultima release del tool SpotBugs.

Al termine dell'analisi viene prodotto un report nella directory "spotbugs-reports" per ogni modulo analizzato.

L'analisi viene effettuata su tutti i sorgenti descritti nella sezione *Sorgenti soggetti a controllo qualità*.

Gli identificativi dei moduli sono classificati come segue:

- utilità di base: utils-commons, utils-generic-project;
- runtime di GovWay: runtime-message, runtime-core, runtime-protocol, runtime-monitor, runtime-security, runtime-pdd;
- profili di interoperabilità: protocol-as4, protocol-modipa, protocol-sdi, protocol-spcoop, protocol-trasparente;
- console web: web-lib-audit, web-lib-mvc, web-lib-queue, web-lib-users, web-loader, web-govwayConsole, web-govwayMonitor;

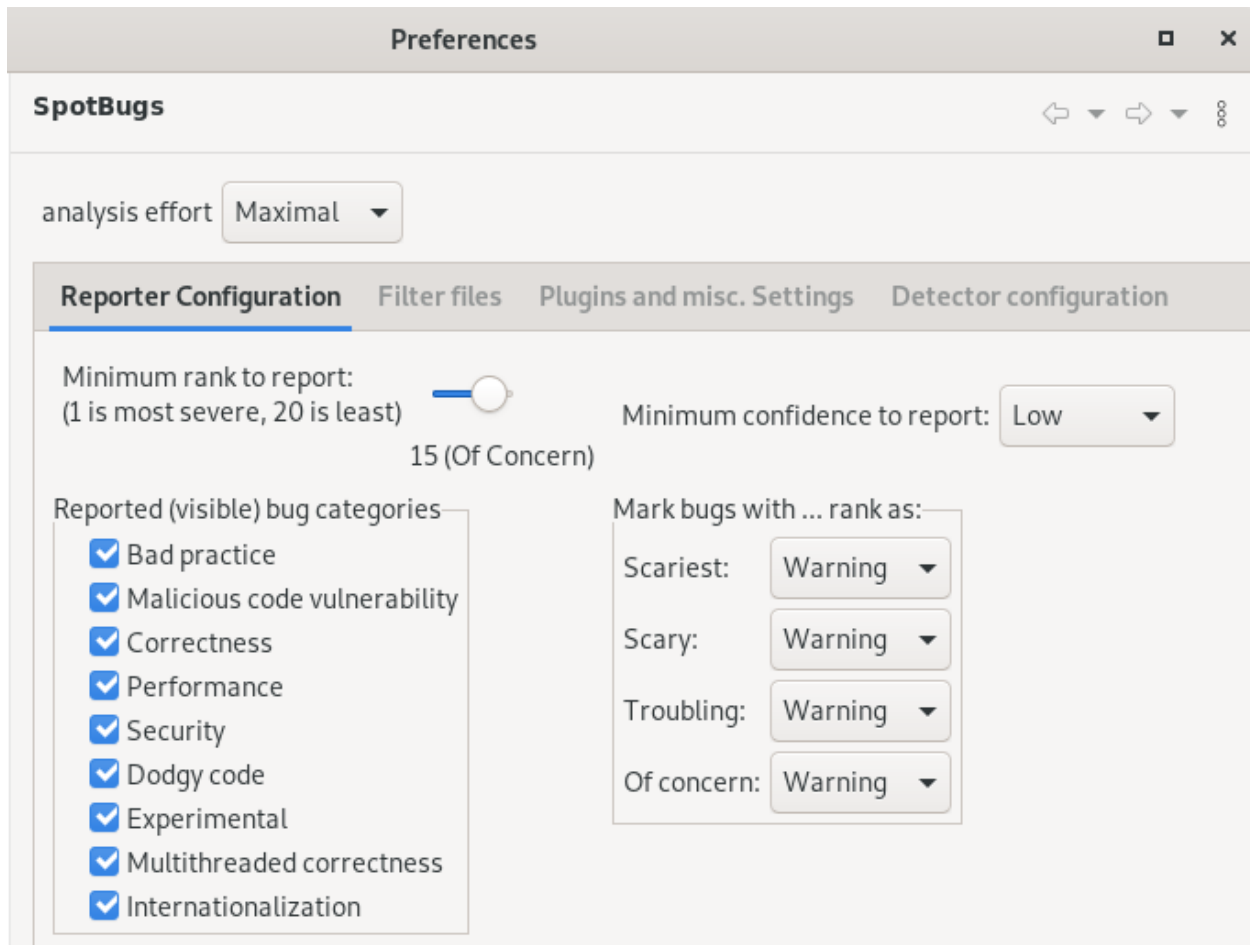


Fig. 2.13: SpotBugs Eclipse Plugin: configurazione

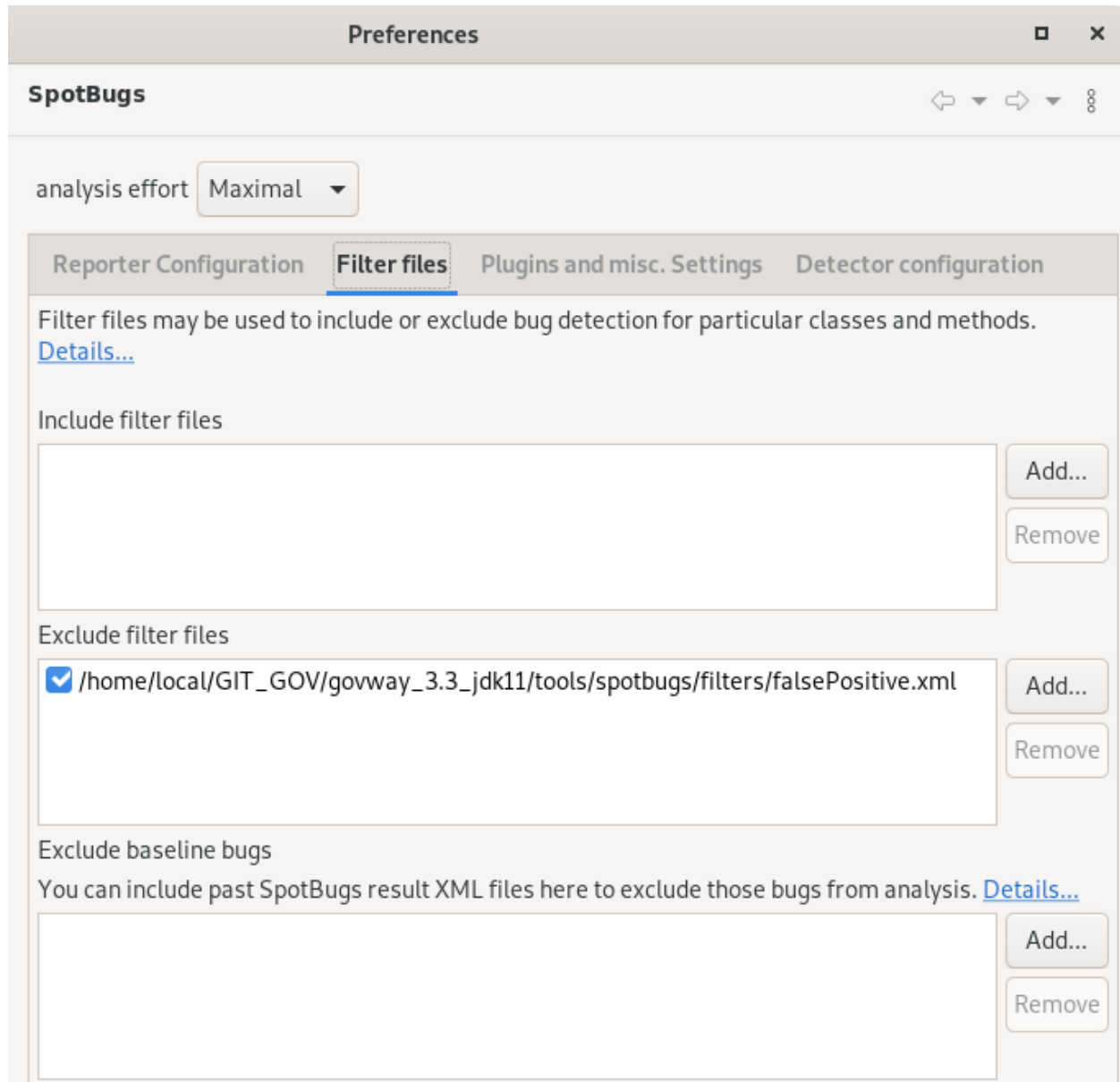


Fig. 2.14: SpotBugs Eclipse Plugin: filtro

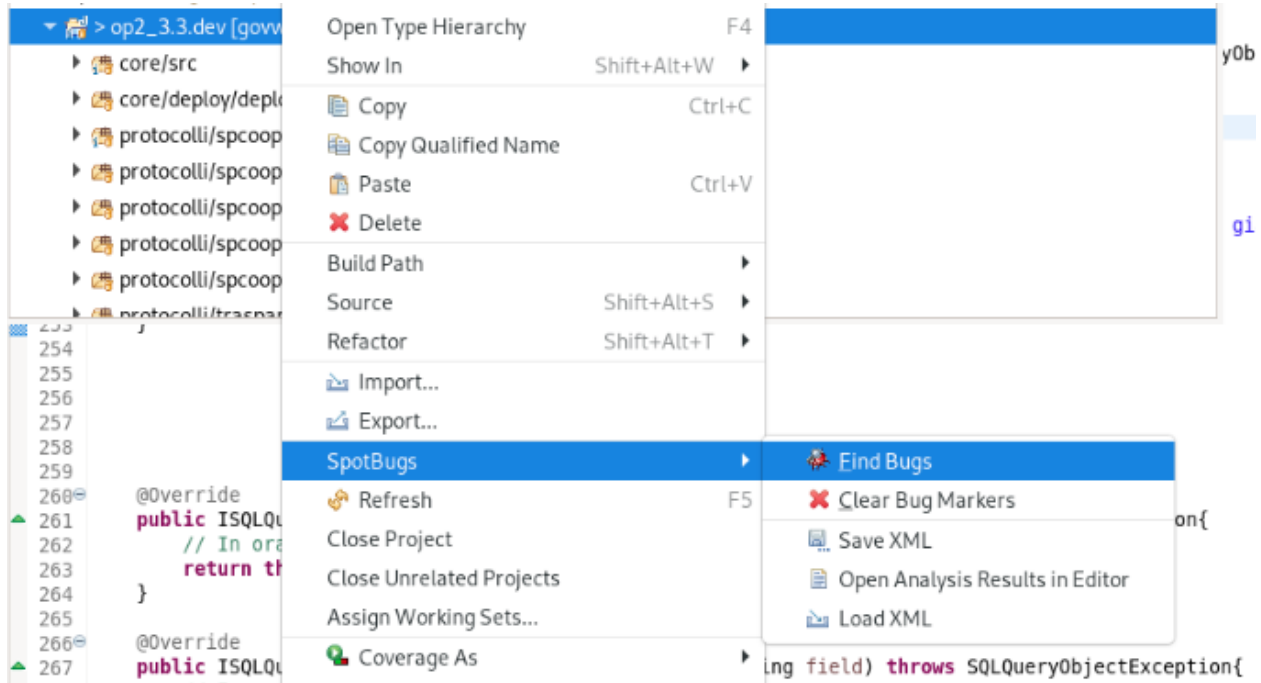


Fig. 2.15: SpotBugs Eclipse Plugin: find bugs

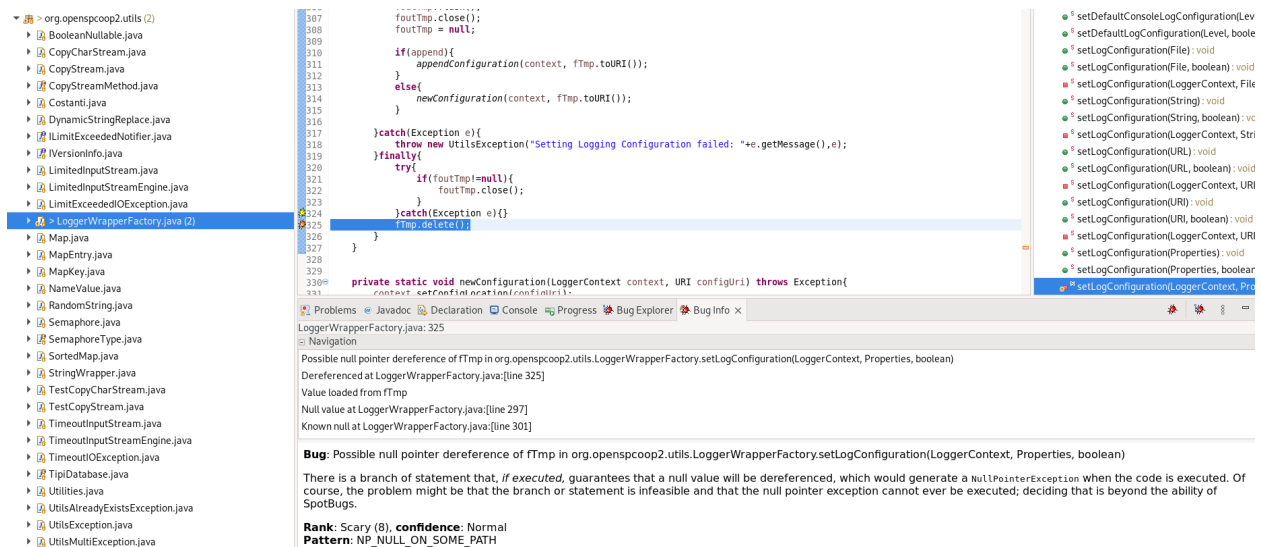
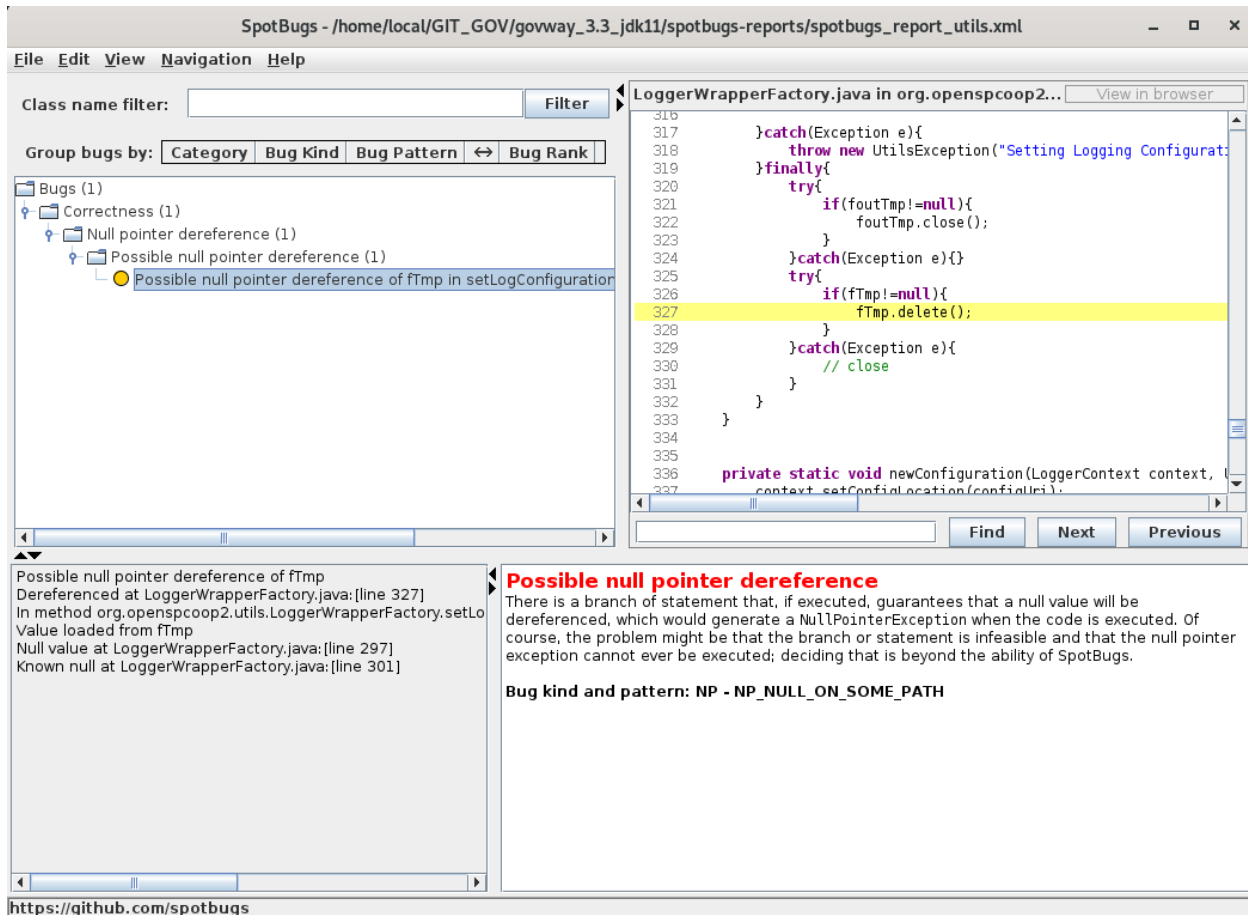


Fig. 2.16: SpotBugs Eclipse Plugin: esempio di bug

- api di configurazione e monitoraggio: rs-config, rs-monitor;
- batch: batch-statistiche, batch-runtime-repository.

I reports prodotti nella directory “spotbugs-reports” sono analizzabili tramite la [SpotBugs GUI](#) avviabile tramite lo script presente in [tools/spotbugs/startSpotBugsAnalysisConsole.sh](#). La figura Fig. ?? mostra un esempio di report analizzato con la console di SpotBugs.



Per produrre un report in un formato direttamente fruibile senza dover utilizzare la [SpotBugs GUI](#) è possibile indicare il formato desiderato tramite il parametro “spotbugs.outputType”.

```
mvn verify -Dspotbugs=verify -Dspotbugs.outputType=html -Dspotbugs.home=PATH_ASSOLUTO_
↳TOOLS_SPOTBUGS -Dtestsuite=none -Dpackage=none -Dwasps=none
```

I formati supportati sono i seguenti:

- xml:withMessages: xml che contiene anche messaggi “human-readable”;
- html: pagina HTML con “default.xml” come stylesheet;
- text: formato testuale;
- emacs: formato “Emacs error message”;
- xdocs: formato xdoc XML da usare con Apache Maven.

Per evitare la verifica di alcuni moduli è possibile utilizzare la proprietà “spotbugs.skipPackages”.

L'esempio seguente attiva l'analisi dei sorgenti solamente per le utilità di base e i componenti di runtime di GovWay:

```
mvn verify -Dspotbugs=verify
           -Dtestsuite=none -Dpackage=none -Dowasp=none
           -Dspotbugs.home=/tmp/spotbugs-4.7.3
           -Dspotbugs.skipPackages=protocol-as4,protocol-modipa,protocol-sdi,protocol-
↪spcoop,protocol-trasparente,web-lib-audit,web-lib-mvc,web-lib-queue,web-lib-users,
↪web-loader,web-govwayConsole,web-govwayMonitor,rs-config,rs-monitor,batch-
↪statistiche,batch-runtime-repository
```

2.3 SonarQube

In questa fase vengono identificate possibili vulnerabilità all'interno del codice sorgente tramite il tool **SonarQube**.

Il tool viene utilizzato fin dalle fasi di sviluppo prima di integrare una nuova funzionalità o un fix di un bug sul master del progetto, verificando la qualità dei sorgenti tramite il **plugin per Eclipse** come descritto nella sezione *SonarLint Eclipse Plugin*.

Ad ogni commit sul **master dei sorgenti del progetto** viene effettuata automaticamente una verifica dei sorgenti nell'ambiente di **Continuous Integration Jenkins di GovWay**. Maggiori dettagli vengono forniti nella sezione *SonarQube Jenkins Plugin*.

Una verifica manuale dei **sorgenti del progetto GovWay** può essere effettuata seguendo le indicazioni presenti nella sezione *SonarQube Maven Plugin*.

2.3.1 SonarQube Jenkins Plugin

Ad ogni commit sul **master dei sorgenti del progetto** viene effettuata automaticamente una verifica dei sorgenti nell'ambiente di **Continuous Integration Jenkins di GovWay**.

L'icona **Fig. 2.17** certifica l'avvenuta integrazione con **SonarQube disponibile nell'ambiente jenkins**.

Per consultare il report relativo all'ultima scansione effettuata sui sorgenti del progetto è possibile collegarsi alla console di **SonarQube dedicato all'ambiente jenkins** con utenza e password "govway". Una volta effettuato il login sarà disponibile tra i progetti analizzati il progetto "govway" (**Fig. 2.18**). Accedendo ai dettagli del progetto si potrà esaminare il report sulla qualità dei sorgenti relativi all'ultimo commit e globale del progetto (**Fig. 2.19**).

2.3.2 SonarLint Eclipse Plugin

In questa sezione viene descritto come utilizzare il **plugin per Eclipse** per la verifica del codice sorgente.

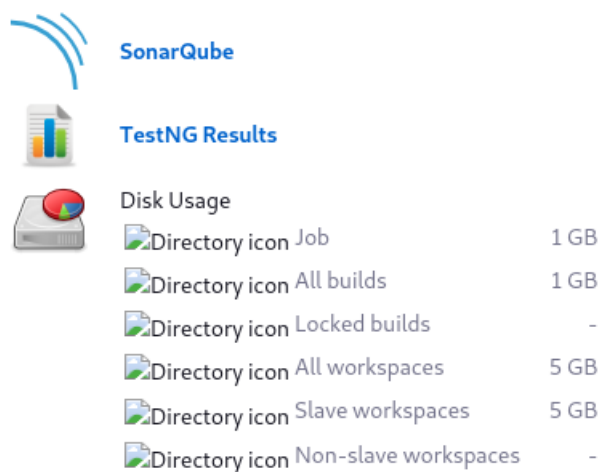
Come prerequisito il plugin deve essere stato installato tramite "Eclipse Marketplace" come mostrato nella figura **Fig. 2.20**.

Deve poi essere configurato il tool **nodejs** versione 16 o 18 accedendo alla sezione «Window -> Preferences -> SonarLint», come mostrato nella figura **Fig. 2.21**:

Il progetto di Eclipse contenente i sorgenti soggetti a controllo della qualità (*Eclipse Project*) deve essere legato a SonarQube effettuando il bind come mostrato nella figure **Fig. 2.22**, **Fig. 2.23**, **Fig. 2.24**, **Fig. 2.25**, **Fig. 2.26**.

L'analisi statica dei sorgenti è adesso effettuabile selezionando il progetto "op2_3.x.dev" (*Eclipse Project*) con il tasto destro e cliccando sulla voce "SonarLint -> Analyze" come mostrato nella figura **Fig. 2.27**.

Eventuali bug individuati vengono evidenziati sulla singola classe come ad esempio viene mostrato nella figura **Fig. 2.28**.



SonarQube Quality Gate

govway **Passed**
 server-side processing: **Success**

Fig. 2.17: SonarQube: integrazione con jenkins

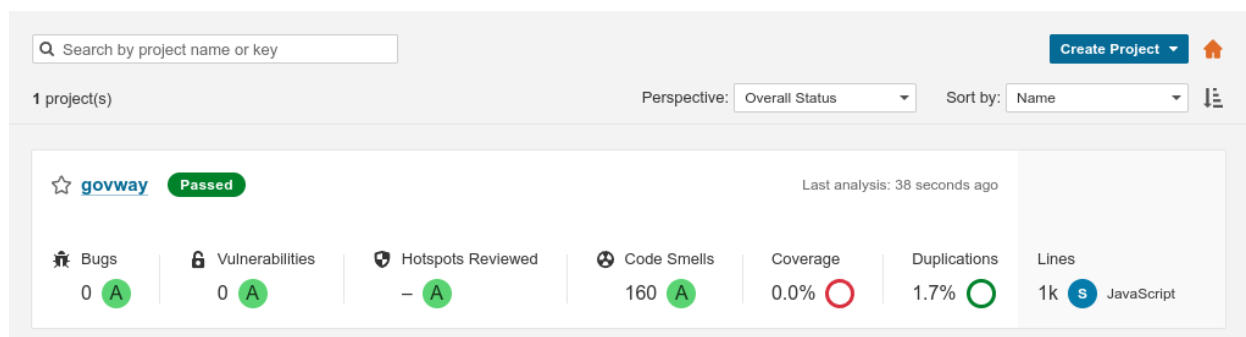


Fig. 2.18: SonarQube: console

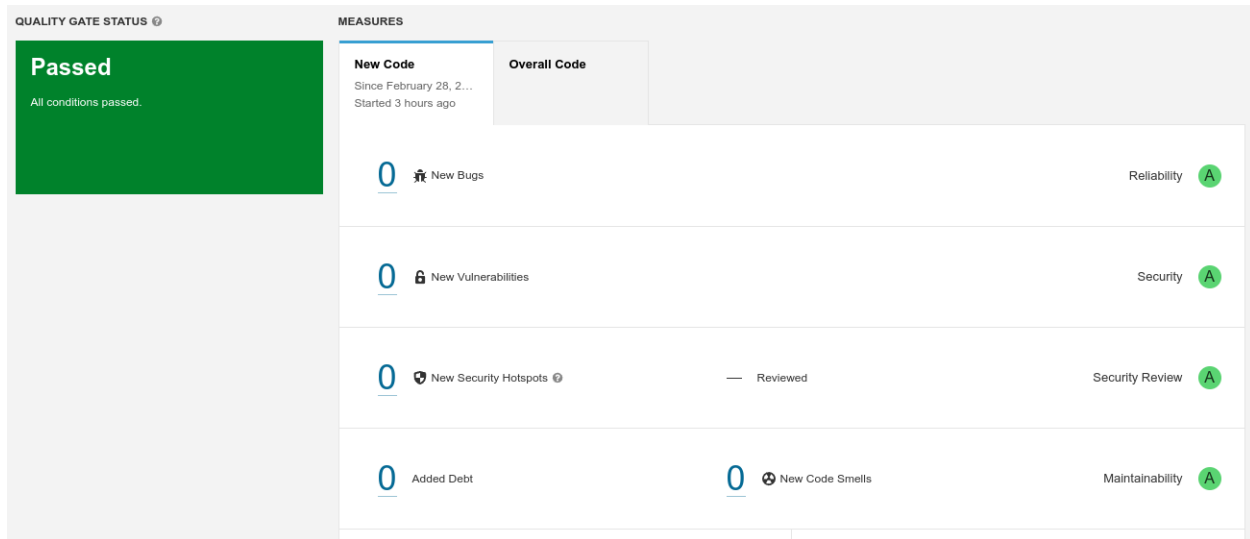


Fig. 2.19: SonarQube: dettagli sulla qualità del software

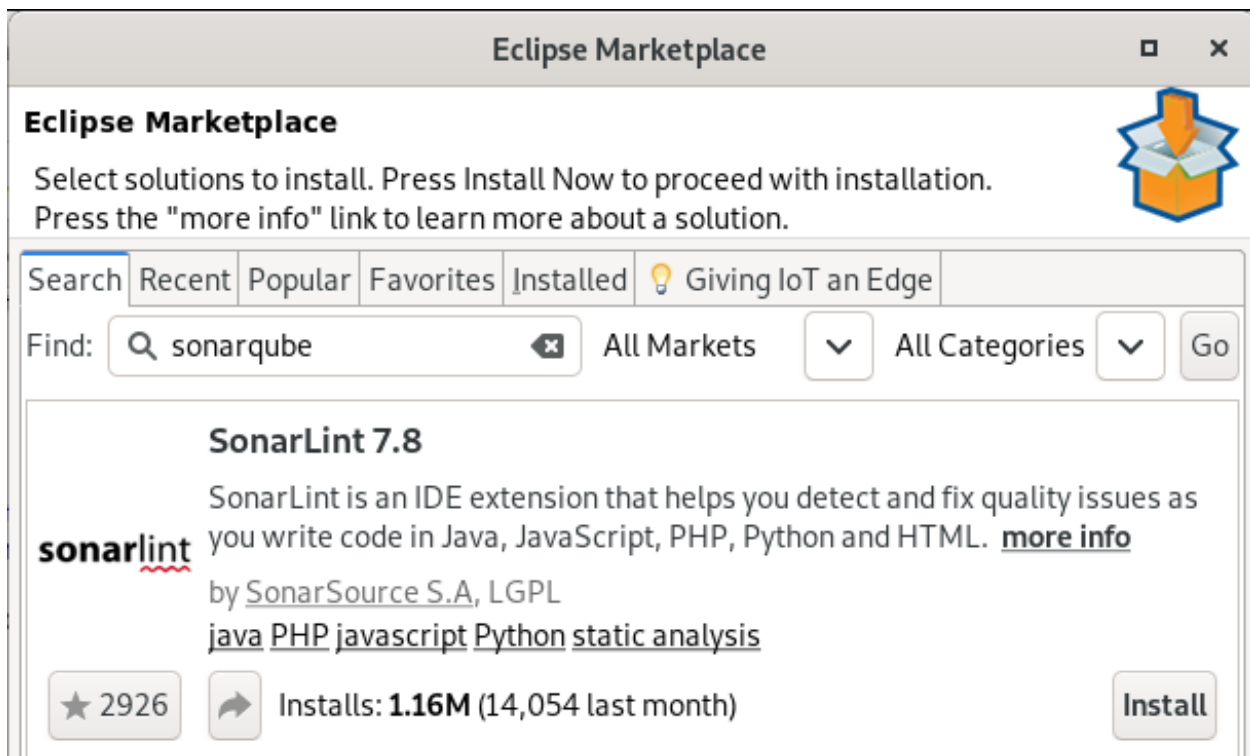


Fig. 2.20: SonarLint Eclipse Plugin: marketplace

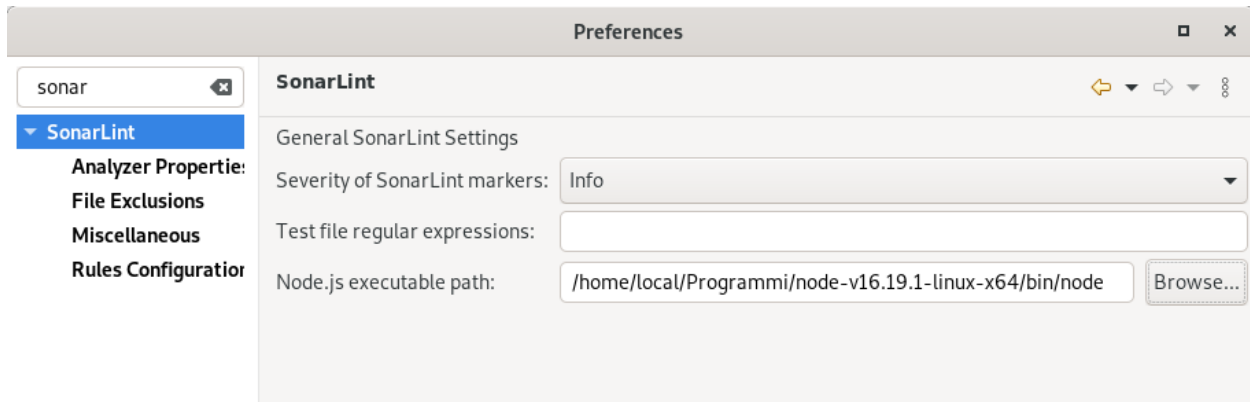


Fig. 2.21: SonarLint Eclipse Plugin: configurazione nodejs



Fig. 2.22: SonarLint Eclipse Plugin: avvio fase di binding

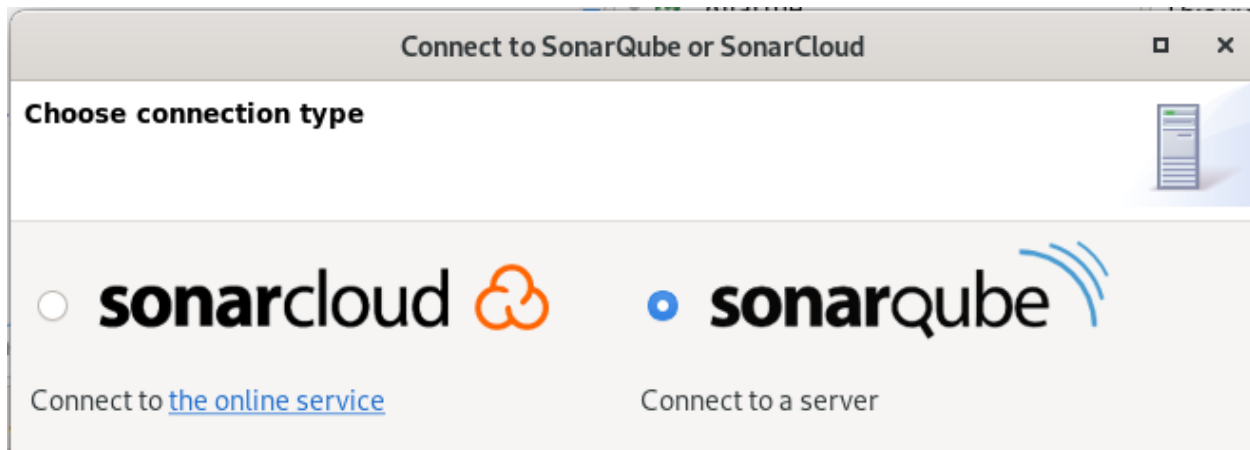


Fig. 2.23: SonarLint Eclipse Plugin: binding verso SonarQube

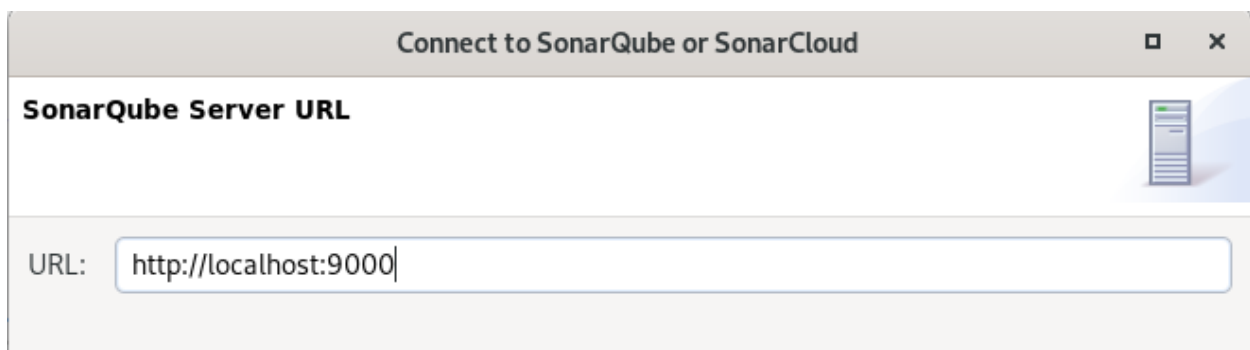


Fig. 2.24: SonarLint Eclipse Plugin: binding url

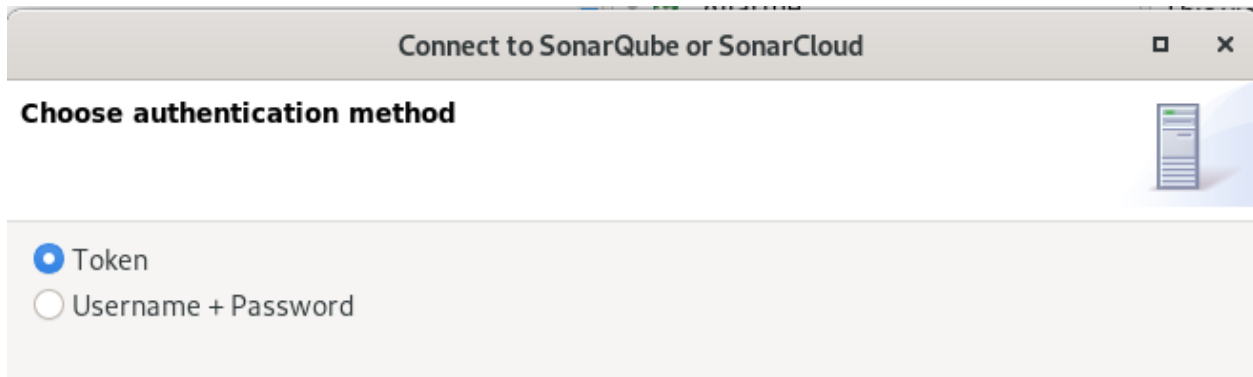


Fig. 2.25: SonarLint Eclipse Plugin: binding authentication type

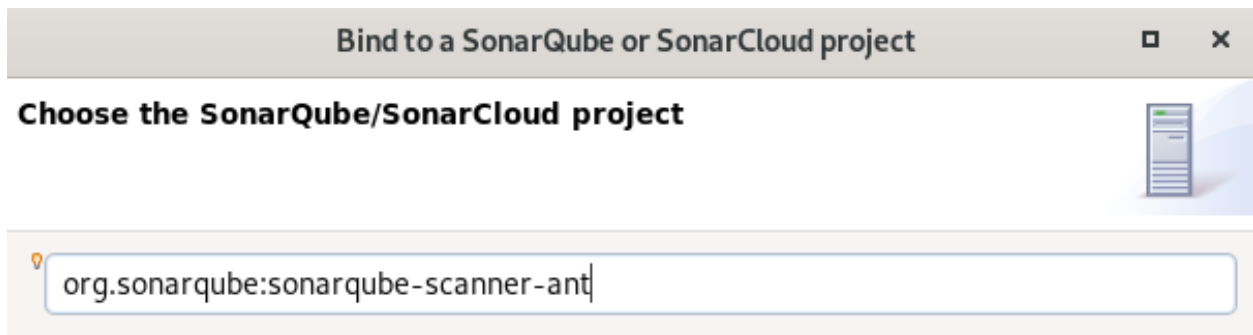


Fig. 2.26: SonarLint Eclipse Plugin: binding project “govway”

2.3.3 SonarQube Maven Plugin

Effettuato il checkout dei [dei sorgenti del progetto GovWay](#), è possibile avviare manualmente l’analisi statica del codice utilizzando il seguente comando maven nella radice del progetto:

```
mvn verify -Dsonarqube=verify -Dsonar.token=TOKEN -Dsonar.host.url=http://
localhost:9000 -Dcompile=none -Dtestsuite=none -Dpackage=none -Dowasp=none
```

Al comando devono essere forniti i parametri per l’accesso all’ambiente SonarQube che si possiede:

- sonar.host.url: indirizzo su cui è in ascolto SonarQube
- sonar.token: token di autenticazione

Al termine dell’analisi viene pubblicato su SonarQube, all’interno del progetto “govway”, un report consultabile all’indirizzo indicato nella proprietà “sonar.host.url”.

L’analisi viene effettuata su tutti i sorgenti descritti nella sezione *Sorgenti soggetti a controllo qualità*.

È possibile effettuare l’analisi di un solo modulo fornendone l’identificativo tramite la proprietà “sonar.govway.project”. Gli identificativi dei moduli analizzati sono i seguenti:

- utilità di base: utils-commons, utils-generic-project;
- runtime di GovWay: runtime-message, runtime-core, runtime-protocol, runtime-monitor, runtime-security, runtime-pdd;
- profili di interoperabilità: protocol-as4, protocol-modipa, protocol-sdi, protocol-spcoop, protocol-trasparente;

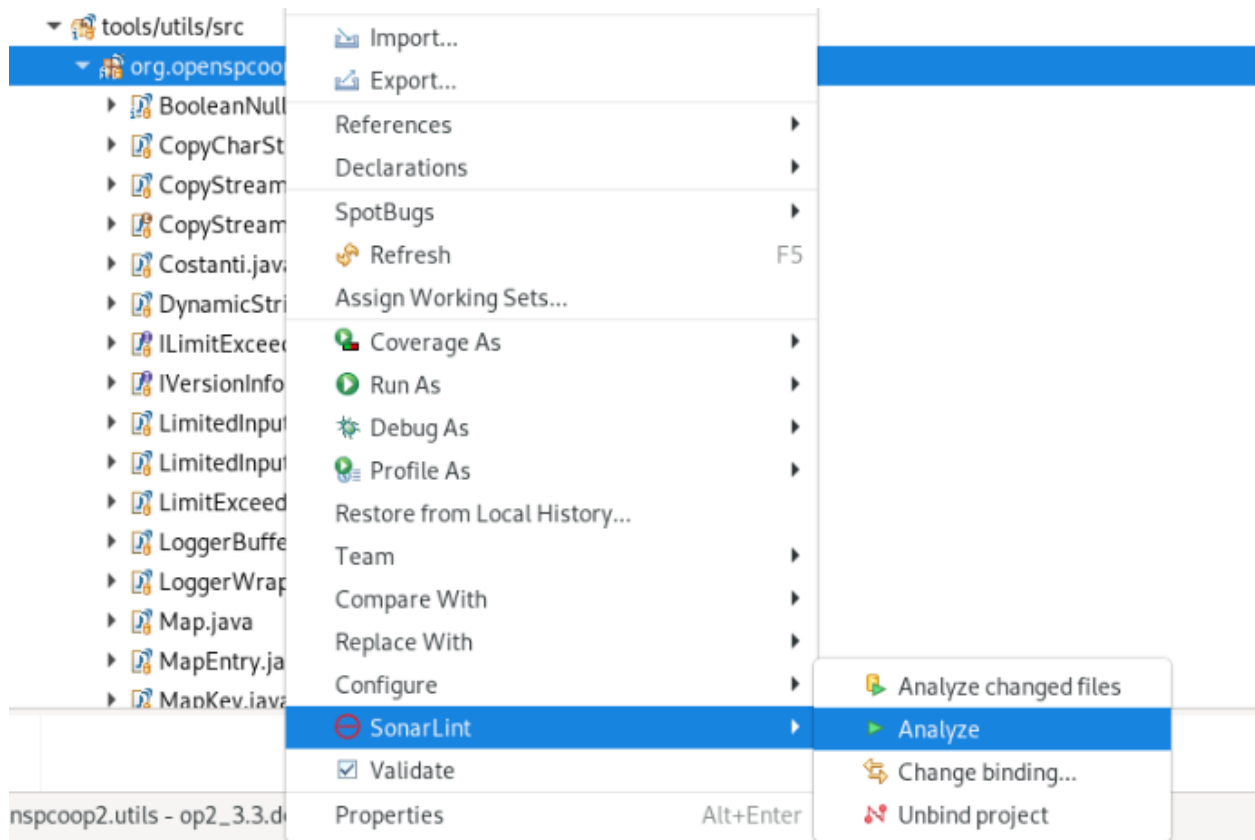


Fig. 2.27: SonarLint Eclipse Plugin: analyze



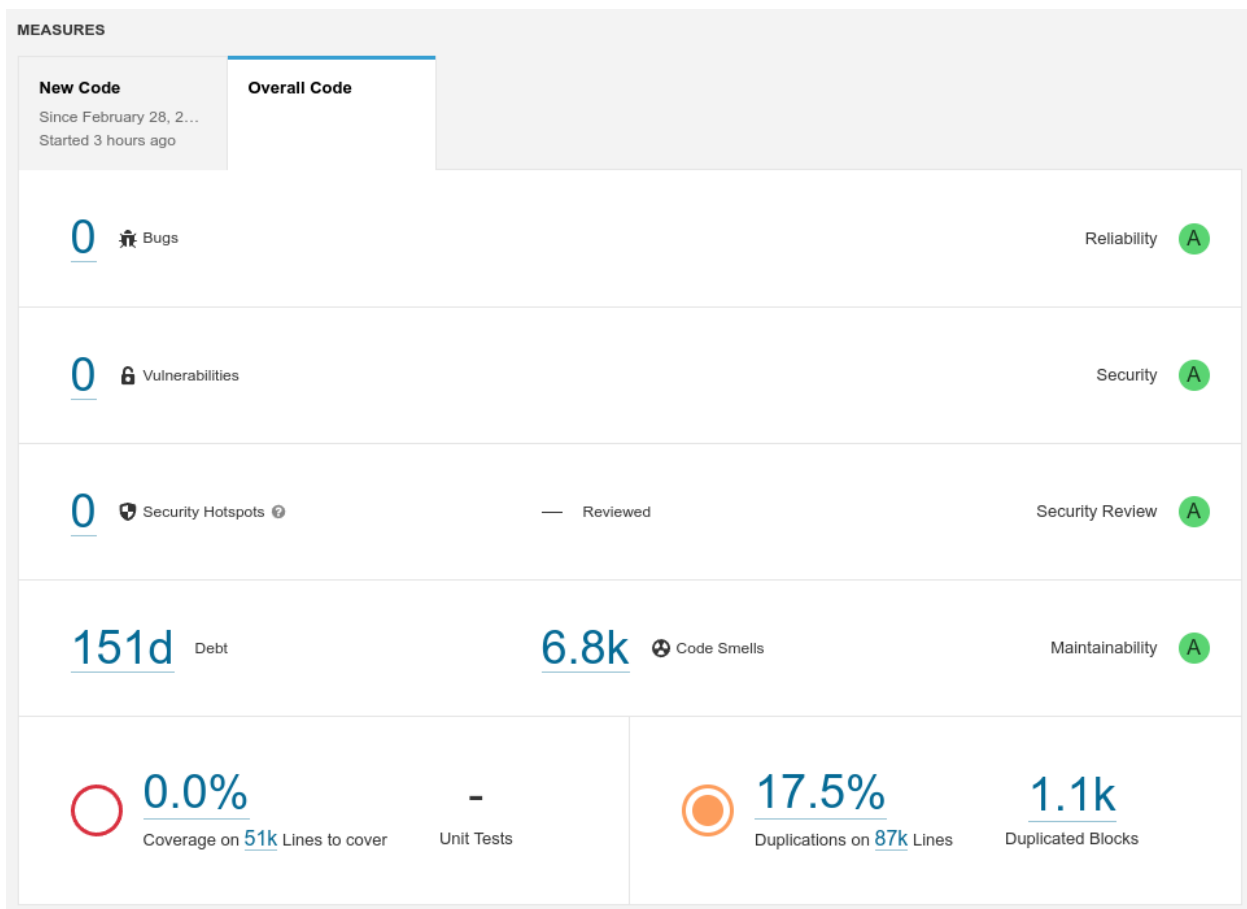
Fig. 2.28: SonarLint Eclipse Plugin: esempio di bug

- console web: web-lib-audit, web-lib-mvc, web-lib-queue, web-lib-users, web-loader, web-govwayConsole, web-govwayMonitor;
- api di configurazione e monitoraggio: rs-config, rs-monitor;
- batch: batch-statistiche, batch-runtime-repository.

L'esempio seguente attiva l'analisi dei sorgenti solamente per il profilo di interoperabilità "ModI":

```
mvn verify -Dsonarqube=verify
           -Dtestsuite=none -Dpackage=none -Dowasp=none
           -Dspotbugs.home=/tmp/spotbugs-4.7.3
           -Dsonar.token=TOKEN -Dsonar.host.url=http://localhost:9000
           -Dsonar.govway.project=protocol-modipa
```

La figura Fig. ?? mostra un esempio di report.



Dynamic Analysis

L'analisi dinamica cerca vulnerabilità del software durante l'effettiva esecuzione del prodotto. L'analisi viene effettuata ad ogni commit sul [master dei sorgenti del progetto](#) nell'ambiente di [Continuous Integration Jenkins di GovWay](#) dove vengono avviati oltre 8.700 test realizzati con il tool [TestNG](#) e oltre 5.300 test realizzati tramite i tool [JUnit](#) e [Karate](#).

All'interno degli oltre 13.000 test, quelli mirati agli aspetti di sicurezza vengono descritti nella sezione [Security Tests](#).

Ulteriori test mirati agli aspetti di sicurezza vengono effettuati utilizzando il tool [OWASP ZAP Proxy](#) per verificare sia il componente runtime di GovWay che le API REST di configurazione e monitoraggio come descritto nella sezione [OWASP Zed Attack Proxy \(ZAP\)](#).

Maggiori dettagli sui test funzionali vengono invece forniti nella sezione [Functional tests](#).

3.1 Security Tests

All'interno degli oltre 13.000 test descritti nella sezione [Functional tests](#) risiedono test dedicati alla sicurezza che vengono descritti nei paragrafi seguenti.

Ulteriori test mirati agli aspetti di sicurezza vengono effettuati utilizzando il tool [OWASP ZAP Proxy](#) descritto nella sezione [OWASP Zed Attack Proxy \(ZAP\)](#).

Se i test rilevano un nuovo problema o una regressione viene avviata una gestione della vulnerabilità come descritto nel manuale [vulnerabilityManagement](#).

3.1.1 Autenticazione

I test realizzati tramite il tool **TestNG** verificano le autenticazioni https, http-basic, principal e api-key.

I sorgenti sono disponibili in `protocolli/trasparente/testsuite/src` relativamente ai seguenti package:

- `org.openspcoop2.protocol.trasparente.testsuite.units.soap.authn`
- `org.openspcoop2.protocol.trasparente.testsuite.units.rest.auth`

Evidenze disponibili in:

- Autenticazione per le API Fruite
- Autenticazione per le API Erogare

Ulteriori evidenze dei test mirati alle sole API di tipo REST che verificano la gestione di header HTTP quali “WWW-Authenticate”, “Authorization” etc sono disponibili in:

- Autenticazione per le API REST Fruite
- Autenticazione per le API REST Erogare.

Altri test vengono realizzati tramite il tool **JUnit** e verificano le autenticazioni tramite token OAuth2, e il forward delle credenziali tramite header HTTP dove l'autenticazione viene effettuata da un webserver. Vengono inoltre verificate le autenticazioni https, http-basic, principal e api-key già verificate tramite TestNG relativamente ai soli applicativi di dominio esterno.

I sorgenti sono disponibili in `protocolli/trasparente/testsuite/karate/src` relativamente ai seguenti package:

- `org.openspcoop2.core.protocolli.trasparente.testsuite.autenticazione.applicativi_token`
- `org.openspcoop2.core.protocolli.trasparente.testsuite.autenticazione.applicativi_esterni`
- `org.openspcoop2.core.protocolli.trasparente.testsuite.autenticazione.gestore_credenziali`
- `org.openspcoop2.core.protocolli.trasparente.testsuite.autenticazione.gestore_credenziali_principal`

Evidenze disponibili in:

- Autenticazione tramite token OAuth2
- Autenticazione applicativi di dominio esterno
- Forward delle credenziali
- Forward delle credenziali “principal”

3.1.2 Autorizzazione

I test realizzati tramite il tool **TestNG** verificano i criteri di autorizzazione puntuale e per ruolo.

I sorgenti sono disponibili in `protocolli/trasparente/testsuite/src` relativamente ai seguenti package:

- `org.openspcoop2.protocol.trasparente.testsuite.units.soap.authz`

Evidenze disponibili in:

- Autorizzazione per le API Fruite
- Autorizzazione per le API Erogatore

3.1.3 Pattern di Sicurezza ModI

I test sono realizzati tramite il tool [Karate](#) e verificano tutti i pattern di sicurezza previsti dalla Linee Guida di Interoperabilità ModI.

I sorgenti sono disponibili in [protocolli/modipa/testsuite/src](#) relativamente ai seguenti package:

- [org.openspcoop2.core.protocolli.modipa.testsuite.rest.sicurezza_messaggio](#); vengono verificati tutti i pattern di sicurezza previsti su API REST:
 - ID_AUTH_REST_01
 - ID_AUTH_REST_02
 - INTEGRITY_REST_01 con ID_AUTH_REST_01
 - INTEGRITY_REST_01 con ID_AUTH_REST_02
 - Token “JWT-Signature” personalizzato (es. Custom-JWT-Signature)
 - Identificazione applicativo e autorizzazione tramite token ModI su API REST
 - Identificazione applicativo e autorizzazione tramite token PDND su API REST
 - Identificazione applicativo e autorizzazione tramite token PDND + token Integrity su API REST
 - Negoziazione token PDND
 - Validazione dei token tramite servizio OCSP su API REST
- [org.openspcoop2.core.protocolli.modipa.testsuite.soap.sicurezza_messaggio](#); vengono verificati tutti i pattern di sicurezza previsti su API SOAP:
 - ID_AUTH_SOAP_01
 - ID_AUTH_SOAP_02
 - INTEGRITY_SOAP_01 con ID_AUTH_SOAP_01
 - INTEGRITY_SOAP_01 con ID_AUTH_SOAP_02
 - Identificazione applicativo e autorizzazione tramite token ModI su API SOAP
 - Identificazione applicativo e autorizzazione tramite token PDND su API SOAP
 - Validazione dei token tramite servizio OCSP su API SOAP

Evidenze disponibili in:

- API REST
- API SOAP

3.1.4 Token Policy e Attribute Authority

I test realizzati tramite il tool [JUnit](#) verificano il processo di validazione e autorizzazione dei token OAuth2, quello di negoziazione e quello di scambio di attributi con AttributeAuthority.

I sorgenti sono disponibili in [protocolli/trasparente/testsuite/karate/src](#) relativamente ai seguenti package:

- [org.openspcoop2.core.protocolli.trasparente.testsuite.token.validazione](#); vengono verificate le token policy di validazione dei token.
- [org.openspcoop2.core.protocolli.trasparente.testsuite.token.negoziazione](#); vengono verificate le token policy di negoziazione dei token.

- `org.openspcoop2.core.protocolli.trasparente.testsuite.token.attribute_authority`; verifica l'interazione con le Attribute Authorities.

Evidenze disponibili in:

- Validazione e autorizzazione dei token OAuth2
- Negoziazione dei token OAuth2
- Scambio di attributi con AttributeAuthority

3.1.5 Sicurezza Messaggio

WSSecurity per API SOAP

I test realizzati tramite il tool `TestNG` verificano tutte le funzionalità di cifratura, firma, SAML per WSSecurity su API SOAP.

I sorgenti sono disponibili in `protocolli/spcoop/testsuite/src` relativamente ai seguenti package:

- `org.openspcoop2.protocol.spcoop.testsuite.units.sicurezza`

Evidenze disponibili in:

- `WSSecurity`

JOSE per API REST

I test realizzati tramite il tool `JUnit` verificano tutte le funzionalità di cifratura e firma per JOSE su API REST.

I sorgenti sono disponibili in `protocolli/trasparente/testsuite/src` relativamente ai seguenti package:

- `org.openspcoop2.protocol.trasparente.testsuite.units.rest.jose`

Evidenze disponibili in:

- `JOSE`

3.1.6 Gestione certificati X.509 e keystore JKS, PKCS12, e PKCS11

I test realizzati tramite il tool `JUnit` verificano tutte le utility che consentono di processare certificati X.509, CRL, e keystore nei vari formati JKS, PKCS12 e PKCS11. Per quanto concerne la gestione dei keystore PKCS11 vengono utilizzati token creati con “softhsm”, il simulatore pkcs11 di dnsssec.

I sorgenti sono disponibili in `tools/utls/src/org/openspcoop2/utls/test` relativamente ai seguenti package:

- `org.openspcoop2.utls.test.security`
- `org.openspcoop2.utls.test.certificate`
- `org.openspcoop2.utls.test.crypto`

Evidenze disponibili in:

- `Utilità di base`

Sono inoltre disponibili ulteriori test che verificano tutte le funzionalità di cifratura e firma tramite JOSE su API REST e tramite WSSecurity su API SOAP, la validazione e negoziazione dei token OAuth2, la connessione TLS etc utilizzando keystore di tipo PKCS11.

I sorgenti sono disponibili in `protocolli/trasparente/testsuite/karate/src` relativamente ai seguenti package:

- `org.openspcoop2.core.protocolli.trasparente.testsuite.pkcs11.x509`

Evidenze disponibili in:

- PKCS11

3.1.7 Online Certificate Status Protocol (OCSP)

I test realizzati tramite il tool **JUnit** verificano tutte le utility che consentono di verificare la validità di un certificato tramite il protocollo OCSP definito nel RFC 2560.

I sorgenti sono disponibili nella classe `tools/utis/src/org/openspcoop2/utis/test/certificate/TestOCSP.java`. Evidenze disponibili in:

- Utilità di base

Sono inoltre disponibili ulteriori test che verificano l'utilizzo del protocollo OCSP nelle varie funzionalità di GovWay dove è necessaria una validazione del certificato: connettore https, autenticazione, gestione delle credenziali, validazione token di sicurezza JOSE su API REST e WSSecurity su API SOAP, validazione dei token OAuth2 e risposte JWS di AttributeAuthority.

I sorgenti sono disponibili in `protocolli/trasparente/testsuite/karate/src` relativamente ai seguenti package:

- `org.openspcoop2.core.protocolli.trasparente.testsuite.other.ocsp`

Evidenze disponibili in:

- OCSP

3.1.8 CORS

I test realizzati tramite il tool **JUnit** verificano tutte le funzionalità relative al CORS.

I sorgenti sono disponibili in `protocolli/trasparente/testsuite/src` relativamente ai seguenti package:

- `org.openspcoop2.protocol.trasparente.testsuite.units.rest.cors`
- `org.openspcoop2.protocol.trasparente.testsuite.units.soap.cors`

Evidenze disponibili in:

- CORS

3.1.9 XML eXternal Entity injection (XXE)

I test realizzati tramite il tool **JUnit** verificano che non sia attuabile un attacco XML eXternal Entity injection (XXE).

I sorgenti sono disponibili in `protocolli/trasparente/testsuite/karate/src` relativamente ai seguenti package:

- `org.openspcoop2.core.protocolli.trasparente.testsuite.encoding.dts`

Evidenze disponibili in:

- XXE

3.1.10 OWASP Zed Attack Proxy (ZAP)

Identifica possibili vulnerabilità sia all'interno del componente runtime di GovWay che nelle API REST e le console di configurazione e monitoraggio tramite il tool [OWASP ZAP Proxy](#).

L'analisi viene effettuata ad ogni commit sul [master dei sorgenti del progetto](#) dove viene avviata automaticamente una verifica nell'ambiente di [Continuous Integration Jenkins di GovWay](#). Maggiori dettagli vengono forniti nella sezione [OWASP ZAP Warnings Jenkins Plugin](#).

Effettuato il checkout dei [dei sorgenti del progetto GovWay](#), è possibile anche avviare manualmente una analisi come descritto nella sezione [OWASP ZAP Maven Plugin](#).

OWASP ZAP Warnings Jenkins Plugin

Una analisi effettuata ogni volta che qualcosa viene modificato consente di rilevare immediatamente eventuali vulnerabilità introdotte.

Ad ogni commit sul [master dei sorgenti del progetto](#) viene effettuata automaticamente una verifica dei sorgenti nell'ambiente di [Continuous Integration Jenkins di GovWay](#).

L'analisi produce un [report di dettaglio](#) sulle vulnerabilità trovate. Per ogni vulnerabilità identificata vengono forniti maggiori dettagli come la severità, il tipo e la url dove è stata riscontrata (es. [Fig. 3.1](#)).

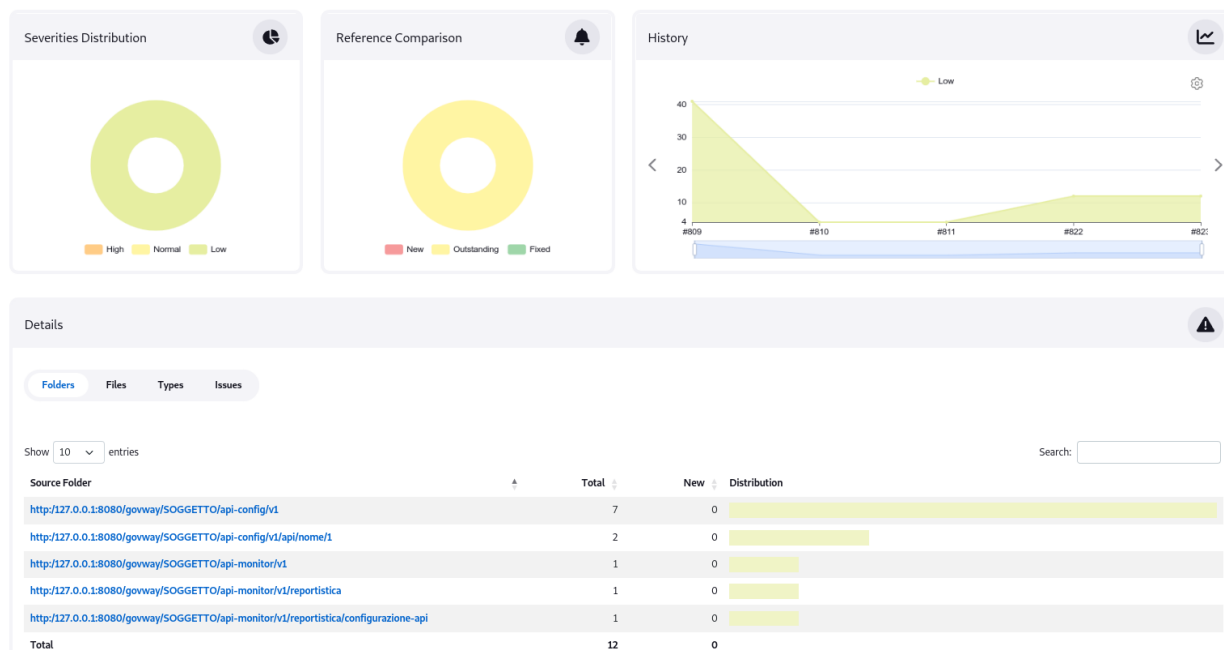


Fig. 3.1: OWASP ZAP: dettaglio di una vulnerabilità

Nella [homepage dell'ambiente CI Jenkins di GovWay](#) è anche disponibile un report che visualizza il trend delle vulnerabilità rispetto ai commit effettuati nel tempo (es. [Fig. 3.2](#)).

Sono inoltre disponibili [report di dettaglio in vari formati](#) ([Fig. 3.3](#)).

La figura [Fig. 3.4](#) mostra un esempio di report nel formato HTML.

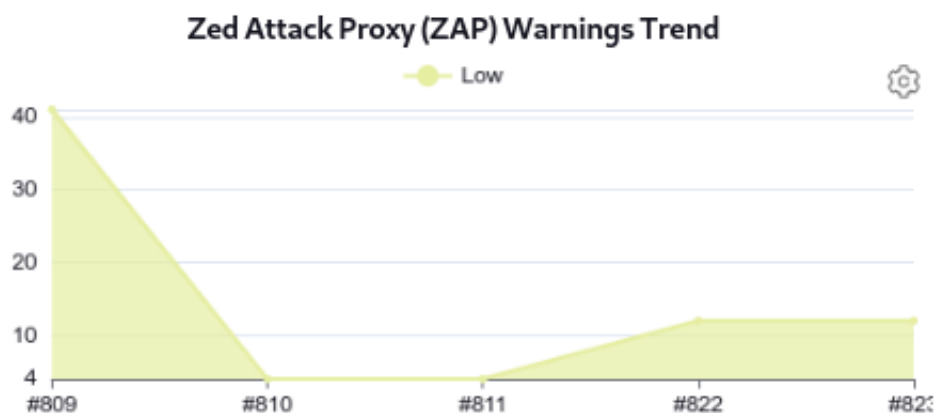


Fig. 3.2: OWASP ZAP Warnings Trend



Fig. 3.3: OWASP ZAP: report in vari formati

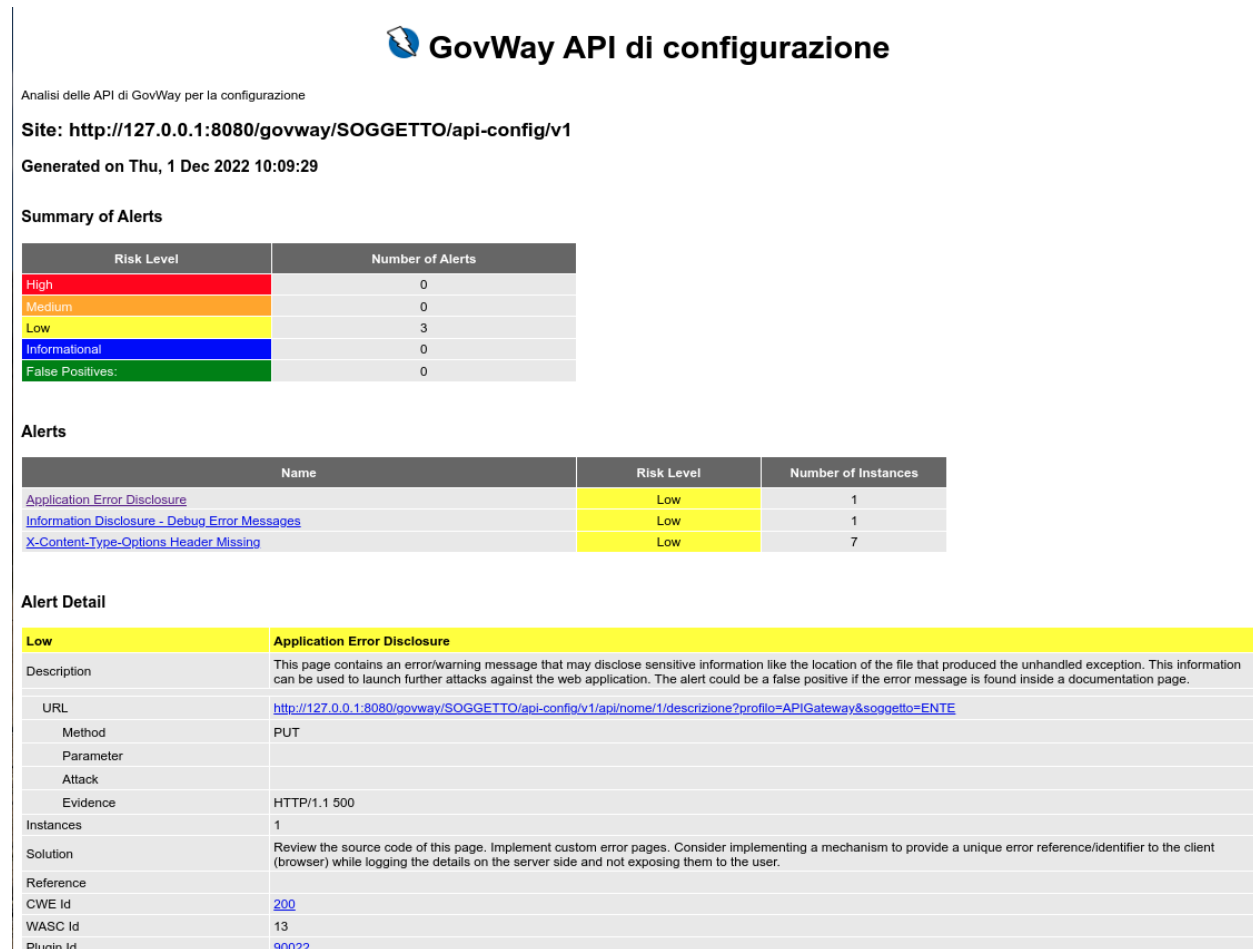


Fig. 3.4: OWASP ZAP: html report

OWASP ZAP Maven Plugin

Effettuato il checkout dei [dei sorgenti del progetto GovWay](#), è possibile avviare manualmente una analisi dinamica, tramite ZAP, utilizzando il seguente comando maven nella radice del progetto:

```
mvn verify -Dzapproxy=verify -Dzapproxy.home=PATH_ASSOLUTO_TOOLS_ZAP -Dgovway.  
↪endpoint=http://127.0.0.1:8080 -Dgovway.ente=<SoggettoIndicatoInstaller> -  
↪Dcompile=none -Dtestsuite=none -Dpackage=none -Dowasp=none
```

Come prerequisito all'esecuzione deve essere effettuato il download dell'ultima release del tool [OWASP ZAP Proxy](#).

Al termine dell'analisi viene prodotto un report nella directory “zapproxy-reports” per ogni componente di GovWay analizzato.

Gli identificativi dei componenti verificati sono i seguenti:

- api-rest-status: vengono verificate le vulnerabilità tipiche per API di tipo REST utilizzando il servizio di health check REST di govway;
- api-soap-status: vengono verificate le vulnerabilità tipiche per API di tipo SOAP utilizzando il servizio di health check SOAP di govway;
- api-config: viene verificata l'API di GovWay per la configurazione;
- api-monitor: viene verificata l'API di GoWay per il monitoraggio.

Per evitare la verifica di alcuni componenti è possibile utilizzare la proprietà “zapproxy.skipTests”.

L'esempio seguente attiva l'analisi solamente del componente runtime di GovWay per API REST e SOAP, escludendo gli altri test:

```
mvn verify -Dzapproxy=verify  
-Dcompile=none -Dtestsuite=none -Dpackage=none -Dowasp=none  
-Dzapproxy.home=/tmp/ZAP_2.12.0  
-Dgovway.endpoint=http://127.0.0.1:8080 -Dgovway.ente=Ente  
-Dzapproxy.skipTests=api-config,api-monitor
```

3.2 Functional tests

Ad ogni commit sul [master dei sorgenti del progetto](#) vengono avviati test mirati ad identificare problematiche e vulnerabilità del software.

Vengono eseguiti oltre 8.700 test realizzati con il tool [TestNG](#) ed oltre 5.300 test realizzati tramite i tool [JUnit](#) e [Karate](#) i cui sorgenti sono disponibili pubblicamente sul [repository dei sorgenti del progetto](#) nei seguenti path:

- test che verificano le utilità di base del progetto (certificati, firma, cifratura ...) risiedono in [tools/utls/src/org/openspcoop2/utls/test](#) e [core/src/org/openspcoop2/pdd_test](#);
- test mirati a verificare svariate funzionalità utilizzando il profilo di interoperabilità “API Gateway” sono disponibili in [protocolli/trasparente/testsuite](#) e [protocolli/trasparente/testsuite/karate](#);
- test che verificando il profilo di interoperabilità “ModI” sono presenti in [protocolli/modipa/testsuite](#);
- test che verificando il profilo di interoperabilità “SPCoop” risiedono in [protocolli/spcoop/testsuite](#);
- test delle API di configurazione disponibili in [tools/rs/config/server/testsuite](#);
- test delle API di monitoraggio disponibili in [tools/rs/monitor/server/testsuite](#).

TestNG Results

2 failures(+2)

8741 tests(±0)

Failed Tests

hide/expand the table

Test Method	Duration
>>> org.openspcoop2.protocol.spcoop.testsuite.units.tunnel_soap.TunnelSOAP.testSincronoMultipartRelatedMIME	00:00:03.061
>>> org.openspcoop2.protocol.spcoop.testsuite.units.tunnel_soap.TunnelSOAP.testSincronoMultipartRelatedMIME	00:00:03.027

All Tests (grouped by their packages)

hide/expand the table

Package	Duration	Fail	(diff)	Skip	(diff)	Total	(diff)
org.openspcoop2.utils.test.regex	00:00:00.003	0	0	0	0	1	0
org.openspcoop2.protocol.trasparente.testsuite.units.rest.method.delete	00:05:23.253	0	0	0	0	302	0
org.openspcoop2.utils.test.openapi	00:01:09.118	0	0	0	0	29	0
org.openspcoop2.pdd_test.dynamic	00:00:03.441	0	0	0	0	1	0
org.openspcoop2.protocol.trasparente.testsuite.units.rest.method.get	00:03:01.155	0	0	0	0	172	0
org.openspcoop2.utils.test.json	00:02:31.551	0	0	0	0	4	0
org.openspcoop2.utils.test.csv	00:00:00.070	0	0	0	0	1	0
org.openspcoop2.protocol.trasparente.testsuite.units.soap.authn	00:02:59.299	0	0	0	0	664	0
org.openspcoop2.protocol.spcoop.testsuite.units.sicurezza	00:04:30.025	0	0	0	0	195	0
org.openspcoop2.protocol.spcoop.testsuite.units.profilo_linee_guida	00:01:02.026	0	0	0	0	158	0

Fig. 3.5: TestNG: dettagli dei test

Test Result

errori (-1)

test (±0)

Durata 5 h 17 min.

Tutti i test falliti

Nome test	Durata
+ org.openscoop2.core.protocolli.trasparente.testsuite.connettori.consegna_con_notifiche.SoapNotificheRisposteTest.notificaRichiesteRisposteSoap12	15 s

Tutti i test

Package	Durata	Falliti	(diff)	Saltati	(diff)
(root)	3 h 0 min	0		0	
org.openscoop2.core.protocolli.trasparente.testsuite.autenticazione applicativi_esterni	2.8 s	0		0	
org.openscoop2.core.protocolli.trasparente.testsuite.autenticazione applicativi_token	50 s	0		0	
org.openscoop2.core.protocolli.trasparente.testsuite.autenticazione.gestore_credenziali	10 s	0		0	
org.openscoop2.core.protocolli.trasparente.testsuite.autenticazione.gestore_credenziali_principal	7.9 s	0		0	
org.openscoop2.core.protocolli.trasparente.testsuite.connettori applicativo_server	9 s	0		0	

Fig. 3.6: JUnit: dettagli dei test

L'analisi produce un report di dettaglio TestNG e un report di dettaglio JUnit che si differenzia per il tool di test utilizzato (es. Fig. 3.5 e Fig. 3.6).

Nella [homepage](#) dell'ambiente CI Jenkins di GovWay è anche disponibile un report che visualizza il trend delle problematiche rilevate rispetto ai commit effettuati nel tempo (es. Fig. 3.7 e Fig. 3.8).

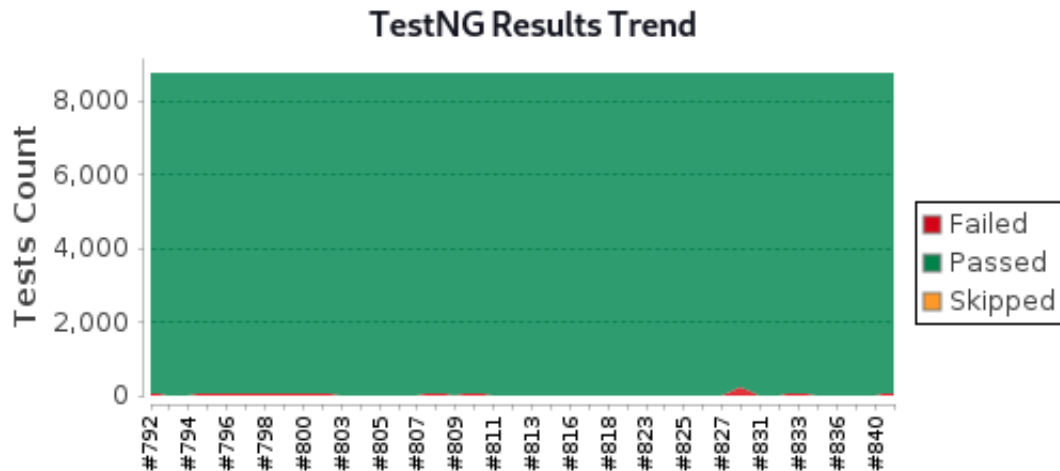


Fig. 3.7: TestNG Results Trend

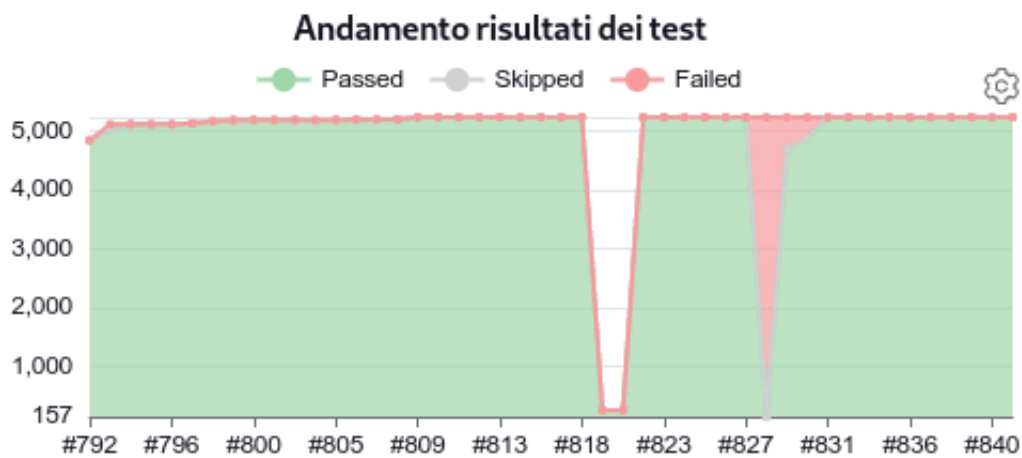


Fig. 3.8: JUnit Results Trend

Sono inoltre disponibili report di dettaglio in vari formati (Fig. 3.9).

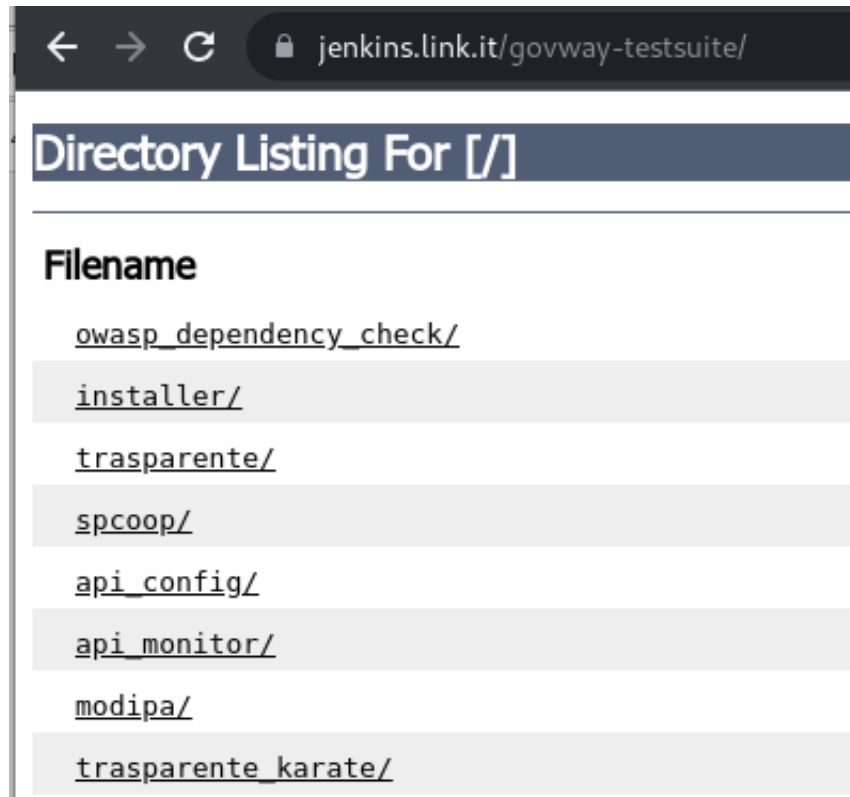


Fig. 3.9: Tests: report in vari formati

3.2.1 Utility di base del progetto

I test sono realizzati tramite il tool JUnit e verificano le utility di base del progetto (certificati, firma, cifratura ...).

I sorgenti sono disponibili in `tools/utls/src/org/openspcoop2/utls/test` relativamente ai seguenti package:

- `org.openspcoop2.utls.test.cache`; vengono verificate le varie implementazioni di cache disponibili.
- `org.openspcoop2.utls.test.certificate`; vengono verificate le utility che consentono di caricare e analizzare key-store (PKCS12, JKS, PKCS11, JWKs), certificati X.509 (DER, PEM) e JWK e protocolli di verifica dei certificati X.509 (CRL e OCSP).
- `org.openspcoop2.utls.test.crypt`; i test verificano la generazione casuale delle password e i meccanismi di digest.
- `org.openspcoop2.utls.test.csv`; vengono verificate le utility che consentono di leggere file csv.
- `org.openspcoop2.utls.test.date`; i test verificano l'utility di formattazione delle date.
- `org.openspcoop2.utls.test.id`; vengono verificate le varie implementazioni che consentono di generare identificativi univoci (numerici, alfanumerici, uuid).
- `org.openspcoop2.utls.test.jdbc`; vengono verificate le utility che consentono di utilizzare funzionalità JDBC specifiche.
- `org.openspcoop2.utls.test.json`; i test verificano le utility che consentono l'esecuzione di JsonPath e la validazione dei messaggi json tramite json schema.
- `org.openspcoop2.utls.test.logger`; verifica il livello di logger applicativo definito nell'utility.
- `org.openspcoop2.utls.test.openapi`; vengono verificate le utility che consentono di leggere e validare interfacce OpenAPI 3 e Swagger 2.

- [org.openspcoop2.utils.test.pdf](#); vengono verificate le utility che consentono di firmare documenti PDF e accedere ai documenti “embedded” o “XFA” interni al pdf stesso.
- [org.openspcoop2.utils.test.random](#); i test verificano l’utility che consente di avere implementazioni differenti di generatori di numeri casuali (NativePRNG, SHA1PRNG, Windows-PRNG ...).
- [org.openspcoop2.utils.test.regex](#); viene verificata l’utility che consente l’applicazione di espressioni regolari.
- [org.openspcoop2.utils.test.resource](#); i test verificano le utility che consentono di lavorare sugli stream.
- [org.openspcoop2.utils.test.rest](#); vengono verificate utility utilizzabili in contesto di API rest (es. Problem Detail RFC 7807).
- [org.openspcoop2.utils.test.security](#); vengono verificate le utility che consentono di firmare/validare o di cifrare/decifrare messaggi. I test prevedono anche la validazione dei certificati X.509 utilizzati tramite CRL e OCSP.
- [org.openspcoop2.utils.test.semaphore](#); viene verificato un semaforo implementato su database.
- [org.openspcoop2.utils.test.serializer](#); i test verificano la serializzazione di java bean.
- [org.openspcoop2.utils.test.sonde](#); i test verificano le utility che consentono di avere sonde applicative.
- [org.openspcoop2.utils.test.sql](#); i test verificano l’utility che consente di definire query SQL tramite oggetti java per poi ottenere lo statement SQL corretto su implementazioni differenti (oracle, postgresql, mysql, hsql, sqlserver, db2).
- [org.openspcoop2.utils.test.transport](#); i test verificano l’utility che consente di ottenere un CORS Filter e l’utility che consente di trattare header HTTP compatibili con RFC 2047.
- [org.openspcoop2.utils.test.wadl](#); vengono verificate le utility che consentono di leggere e validare interfacce WADL.
- [org.openspcoop2.utils.test.xacml](#); i test verificano l’utility che consente di processare policy XACML.
- [org.openspcoop2.utils.test.xml](#); i test verificano la corretta gestione delle entity references, le utility che consentono di processare XQuery, effettuare diff di file xml. Inoltre viene verificato che non sia attuabile un attacco XML eXternal Entity injection (XXE).
- [org.openspcoop2.utils.test.xml2json](#); i test verificano l’utility che consentono la trasformazione di messaggi xml in json e viceversa.

Evidenze disponibili in:

- [Utility di base](#)
- [Utility di base per il database](#)

3.2.2 Messaggi su API REST

I test realizzati tramite il tool [TestNG](#) verificano le normali funzionalità di gateway per API REST, verificando lo scambio di messaggi json, xml, multipart e binari (pdf, doc, zip ...) tramite tutti i possibili metodi http (POST, GET, PUT, DELETE, ...) e i codici di risposta (2xx, 3xx, 4xx e 5xx). I verificano anche il corretto inoltro dell’header http “X-HTTP-Method-Override” durante una POST.

I sorgenti sono disponibili in [protocolli/trasparente/testsuite/src/.../rest/method](#).

Evidenze disponibili in :

- [risultati dei test su API REST per fruizioni](#)
- [risultati dei test su API REST per erogazioni](#)

Sono inoltre disponibili ulteriori test che verificano la corretta gestione dell'header "Content-Type" valorizzato con altri parametri oltre quelli previsti o valorizzato in maniera errata. I sorgenti sono disponibili in [protocolli/trasparente/testsuite/src/.../rest/integrazione](#).

Evidenze disponibili in [risultati dei test su header Content-Type per API REST](#).

3.2.3 Messaggi su API SOAP

I test realizzati tramite il tool [TestNG](#) verificano le normali funzionalità di gateway per API SOAP, verificando comunicazioni con profilo oneway e request-response utilizzando messaggi SOAP 1.1 e 1.2 sia con che senza attachments. Vengono inoltre verificate sia le modalità stateless che la modalità con presa in carico. Infine viene verificata la corretta gestione dei SOAP Fault.

I sorgenti sono disponibili in [protocolli/trasparente/testsuite/src/.../soap/successful](#).

Evidenze disponibili in :

- [risultati dei test su API SOAP per fruizioni](#)
- [risultati dei test su API SOAP per erogazioni](#)

Sono disponibili ulteriori test che verificano le funzionalità SOAP descritte dai seguenti gruppi:

- [SOAPWithAttachments](#); vengono verificati per tutti i profili di interazione (oneway, request-response, async pull o push) con messaggi SOAP With Attachments.
- [SOAPAction](#); viene verificato che il gateway gestisca correttamente le possibili SOAPAction ricevute nell'header di trasporto HTTP.
- [SOAPBodyEmpty](#); viene verificato che il gateway gestisca correttamente messaggi senza SOAPBody (vuoto).
- [SOAPHeaderEmpty](#); viene verificato che il gateway gestisca correttamente messaggi senza SOAPHeader.
- [SOAPMessageScorretti](#); viene verificato che il gateway gestisca correttamente messaggi scorretti sintatticamente o rispetto alla specifica soap (es. ContentType/Namespace diverso da quello atteso, headers non gestiti dalla PdD, strutture xml errate).
- [TunnelSOAP](#); vengono verificate le funzionalità di imbustamento e sbustamento SOAP.

Evidenze disponibili in:

- [risultati dei gruppi "SOAPWithAttachments", "SOAPAction", "SOAPBodyEmpty" e "SOAPHeaderEmpty"](#)
- [risultati del gruppo "SOAPMessageScorretti"](#)
- [risultati del gruppo "TunnelSOAP"](#)

Altri test disponibili verificano la corretta gestione dell'header "Content-Type" valorizzato con altri parametri oltre quelli previsti o valorizzato in maniera errata. I sorgenti sono disponibili in [protocolli/trasparente/testsuite/src/.../soap/integrazione](#).

Evidenze disponibili in [risultati dei test su header Content-Type per API SOAP](#).

Sono infine disponibili ulteriori test che verificano la "funzionalità "SOAPReader" per la lettura ottimizzata dei messaggi soap.

I sorgenti sono disponibili in [core/src/org/openspcoop2/pdd_test/.../message/TestSoapReader.java](#).

Evidenze disponibili in [risultati dei test per la funzionalità "SOAPReader"](#)

3.2.4 Connettore

I test realizzati tramite il tool **JUnit** verificano i connettori disponibili e le funzionalità associate.

I sorgenti sono disponibili in [protocolli/trasparente/testsuite/karate/src](#) relativamente ai seguenti gruppi:

- `connettori.timeout`; vengono verificati i parametri di “connection timeout” e “read timeout” impostati relativi alla connessione e alla lettura dei messaggi di richiesta e risposta.
- `connettori.errori`; verifica che, su API SOAP, eventuali risposte di errore senza un payload o con un payload html vengano gestite correttamente.
- `connettori applicativo_server`; viene verificato il funzionamento degli applicativi di tipo server.
- `connettori.redirect`; viene verificata la funzionalità di “follow redirect” e renaming dell’header Location come “proxy pass”.
- `connettori.consegna_condizionale`; viene verificata la funzionalità dei connettori multipli con tipo “Consegna Condizionale”.
- `connettori.load_balancer`; viene verificata la funzionalità dei connettori multipli con tipo “Load Balancer”.
- `connettori.consegna_multipla`; viene verificata la funzionalità dei connettori multipli con tipo “Consegna Multipla”.
- `connettori.consegna_con_notifiche`; viene verificata la funzionalità dei connettori multipli con tipo “Consegna con Notifiche”.
- `connettori.proxy_pass`; viene verificata la funzionalità relativa alle regole di proxy pass.
- `connettori.override_jvm`; viene verificata la funzionalità che consente di modificare la configurazione jvm utilizzata per endpoint https, consentendo di personalizzare i keystore utilizzati tramite la definizione di un file di proprietà.

Evidenze disponibili in:

- risultati dei test del gruppo “connettori.timeout”
- risultati dei test del gruppo “connettori.errori”
- risultati dei test del gruppo “connettori applicativo_server”
- risultati dei test del gruppo “connettori.redirect”
- risultati dei test del gruppo “connettori.consegna_condizionale”
- risultati dei test del gruppo “connettori.load_balancer”
- risultati dei test del gruppo “connettori.consegna_multipla”
- risultati dei test del gruppo “connettori.consegna_con_notifiche”
- risultati dei test del gruppo “connettori.proxy_pass”
- risultati dei test del gruppo “connettori.override_jvm”

Sono inoltre disponibili ulteriori test realizzati tramite il tool **TestNG** i cui sorgenti sono disponibili in [protocolli/spcoop/testsuite/src](#) relativamente ai seguenti gruppi:

- `ConnettoriDiversiHTTP`; vengono verificati i connettori built-in diversi da http e https (es. JMS, File, null, echo).
- `HTTPS`; verifica il corretto funzionamento del connettore https e dell’autenticazione “tls”.
- `LetturaCredenzialiIngresso`; verifica il meccanismo di plugin per l’implementazione di un gestore delle credenziali.

- `VerificaTimeoutGestioneContentLength`; vengono verificate che le connessioni gestite tramite content-length impostato nell'header http di risposta non provochino attese dovute all'impostazione di un content length errato.
- `Servizio Integration Manager`; viene verificato il servizio "Message Box".
- `UrlPrefixRewriter`; vengono verificate le funzionalità di "pd-url-prefix-rewriter" e "pa-url-prefix-rewriter".

Evidenze disponibili in:

- risultati dei test sui connettori
- risultati dei test sul servizio Integration Manager
- risultati dei test sulla funzionalità di "UrlPrefixRewriter"

3.2.5 Rate Limiting

I test sono realizzati tramite il tool JUnit e verificano tutte le funzionalità relative al controllo del traffico.

I sorgenti sono disponibili in `protocolli/trasparente/testsuite/karate/src` relativamente ai seguenti gruppi:

- `rate_limiting.numero_richieste`; vengono verificate policy definite con metriche basate sul numero di richieste simultanee e richieste gestibili in un intervallo temporale.
- `rate_limiting.numero_richieste_fallite`; vengono verificate policy definite con metriche basate sul numero di richieste fallite in un intervallo temporale.
- `rate_limiting.numero_richieste_fallite_o_fault`; vengono verificate policy definite con metriche basate sul numero di richieste fallite o fault in un intervallo temporale.
- `rate_limiting.numero_richieste_fault`; vengono verificate policy definite con metriche basate sul numero di fault in un intervallo temporale.
- `rate_limiting.numero_richieste_completate_con_successo`; vengono verificate policy definite con metriche basate sul numero di richieste completate con successo in un intervallo temporale.
- `rate_limiting.dimensione_messaggi`; vengono verificate policy definite con metriche basate sulla dimensione massima delle richieste e delle risposte.
- `rate_limiting.occupazione_banda`; vengono verificate policy definite con metriche basate sull'occupazione di banda in un intervallo temporale.
- `rate_limiting.tempo_complessivo_risposta`; vengono verificate policy definite con metriche basate sul tempo complessivo di risposta in un intervallo temporale.
- `rate_limiting.tempo_medio_risposta`; vengono verificate policy definite con metriche basate sul tempo medio di risposta in un intervallo temporale.
- `rate_limiting.filtri`; vengono verificati i filtri associabili alle policy.
- `rate_limiting.raggruppamento`; vengono verificati i criteri di conteggio associabili alle policy.
- `rate_limiting.flusso`; vengono verificati i criteri ordinamento dell'applicabilità delle policy.
- `rate_limiting.warning_only`; vengono verificate policy definite con stato warning_only, controllando anche gli eventi generati con stesso stato.
- `rate_limiting.congestione`; vengono verificate policy che usano l'applicabilità con degrado prestazionale e/o congestione anche rispetto agli eventi generati.
- `rate_limiting.global_policy`; vengono verificate policy definite a livello globale.
- `rate_limiting.custom_policy`; vengono verificate policy che usano intervalli statistici di campionamento.

Evidenze disponibili in:

- risultati dei test del gruppo “rate_limiting.numero_richieste”
- risultati dei test del gruppo “rate_limiting.numero_richieste_fallite”
- risultati dei test del gruppo “rate_limiting.numero_richieste_fallite_o_fault”
- risultati dei test del gruppo “rate_limiting.numero_richieste_fault”
- risultati dei test del gruppo “rate_limiting.numero_richieste_completate_con_successo”
- risultati dei test del gruppo “rate_limiting.dimensione_messaggi per API Rest”
- risultati dei test del gruppo “rate_limiting.dimensione_messaggi per API Soap”
- risultati dei test del gruppo “rate_limiting.occupazione_banda”
- risultati dei test del gruppo “rate_limiting.tempo_complessivo_risposta”
- risultati dei test del gruppo “rate_limiting.tempo_medio_risposta”
- risultati dei test del gruppo “rate_limiting.filtri”
- risultati dei test del gruppo “rate_limiting.raggruppamento”
- risultati dei test del gruppo “rate_limiting.flusso”
- risultati dei test del gruppo “rate_limiting.warning_only”
- risultati dei test del gruppo “rate_limiting.congestione” per API Rest
- risultati dei test del gruppo “rate_limiting.congestione” per API Soap
- risultati dei test del gruppo “rate_limiting.global_policy”
- risultati dei test del gruppo “rate_limiting.custom_policy” per API Rest
- risultati dei test del gruppo “rate_limiting.custom_policy” per API Soap

3.2.6 Validazione dei messaggi

I test realizzati tramite il tool JUnit verificano le funzionalità di validazione dei messaggi tramite interfacce OpenAPI 3, Swagger 2 e WSDL.

I sorgenti sono disponibili in [protocolli/trasparente/testsuite/karate/src](#) relativamente ai seguenti gruppi:

- [validazione.parametri](#); viene verificata la serializzazione dei parametri descritta in [Swagger.io - Parameter Serialization](#). I test verificano anche la possibilità che http header e parametri della url possano esistere molteplici volte nella richiesta e nella risposta.
- [validazione.multipart](#); vengono verificati i messaggi multipart descritti in [Swagger.io - Multipart Requests](#) e [Swagger.io - File Upload](#).
- [validazione.swagger_request_validator](#); viene verificata la funzionalità di validazione dei contenuti attuata tramite la libreria [swagger-request-validator](#). Tra i vari test è presente anche la validazione tramite API complesse di dimensioni notevoli.
- [validazione.rpc](#); verifica il funzionamento con messaggi definiti in interfacce WSDL con “style RPC” e “use literal” o “encoded”. I test oltre alla validazione verificano anche il riconoscimento dell’operazione tramite l’ottimizzazione “soap reader”.
- [other.api_grandi](#); verifica l’utilizzo di API REST con un numero elevato di risorse.

Evidenze disponibili in:

- risultati dei test del gruppo “validazione.parametri”
- risultati dei test del gruppo “validazione.multipart”

- risultati dei test del gruppo “validazione.swagger_request_validator”
- risultati dei test del gruppo “validazione.rpc”
- risultati dei test del gruppo “other.api_grandi”

Sono inoltre disponibili ulteriori test realizzati tramite il tool **TestNG** che verificano la validazione effettuata tramite interfaccia WSDL e schemi XSD di un servizio dummy definito tramite molteplici port-types, operations e stili/usi differenti (wrapped document literal, rpc literal, encoded ...) che consentono di validare la conformità dei messaggi sui vari stili.

I sorgenti sono disponibili in:

- [validazione wsdl](#)

Evidenze disponibili in:

- [risultati dei test di validazione](#)

3.2.7 Caching Risposta

I test realizzati tramite il tool **TestNG** verificano le funzionalità di salvataggio delle risposte in una cache. Le risposte, una volta salvate, saranno ritornate al client nelle successive invocazioni senza coinvolgere il backend.

I sorgenti sono disponibili in [protocolli/trasparente/testsuite/src](#) relativamente ai seguenti gruppi:

- [rest/response_caching](#)
- [soap/response_caching](#)

Evidenze disponibili in [risultati dei test sul Caching della Risposta](#)

3.2.8 Trasformazioni

I test realizzati tramite il tool **JUnit** verificano le funzionalità di trasformazione dei messaggi.

I sorgenti sono disponibili in [protocolli/trasparente/testsuite/karate/src](#) relativamente ai seguenti gruppi:

- [trasformazione.protocollo](#); viene attuata una trasformazione completa dei messaggi (contenuti, header, url) per verificare la funzionalità di trasformazione del protocollo rest(json)->soap(xml) e viceversa.
- [trasformazione.form](#); la trasformazione accede ad una richiesta di tipo “application/x-www-form-urlencoded” al fine per riconoscere casi di errori generati durante la certificazione di uno SPIDProvider.
- [trasformazione.soap_action](#); vengono modificate le SOAPAction su messaggi SOAP.
- [trasformazione.info_integrazione](#); vengono inoltrate al backend informazioni estratte dalla richiesta.

Evidenze disponibili in:

- [risultati dei test del gruppo “trasformazione.protocollo”](#)
- [risultati dei test del gruppo “trasformazione.form”](#)
- [risultati dei test del gruppo “trasformazione.soap_action”](#)
- [risultati dei test del gruppo “trasformazione.info_integrazione”](#)

Sono inoltre disponibili ulteriori test che verificano l'utilizzo di variabili dinamiche risolte a runtime dal gateway.

I sorgenti sono disponibili in [core/src/org/openspcoop2/pdd_test/](#) relativamente ai seguenti package:

- [dynamic](#)
- [trasformazioni](#)

Evidenze disponibili in:

- risultati dei test di trasformazione

3.2.9 MTOM

I test realizzati tramite il tool [TestNG](#) verificano le funzionalità di imbustamento, sbustamento e validazione dei messaggi soap serializzati tramite MTOM.

I sorgenti sono disponibili in [protocolli/trasparente/testsuite/src/.../soap/mtom](#).

Evidenze disponibili in :

- risultati dei test sulla gestione MTOM per le fruizioni
- risultati dei test sulla gestione MTOM per le erogazioni

Sono inoltre disponibili ulteriori test che verifica la gestione del protocollo MTOM.

I sorgenti sono disponibili in [core/src/org/openspcoop2/pdd_test/.../message/TestMTOM.java](#).

Evidenze disponibili in risultati dei test MTOM

3.2.10 Correlazione Applicativa

I test realizzati tramite il tool [TestNG](#) verificano le funzionalità di correlazione applicativa che consente al gateway di estrarre un identificatore relativo al contenuto applicativo e associarlo alla traccia a completamento delle informazioni già presenti.

I sorgenti sono disponibili in [protocolli/spcoop/testsuite/src/.../integrazione/IntegrazioneCorrelazioneApplicativa.java](#).

Evidenze disponibili insieme ai test sugli header di integrazione:

- risultati dei test per la correlazione applicativa

Sono inoltre disponibili ulteriori test realizzati tramite il tool [JUnit](#) già descritti nella sezione [Header di Integrazione](#) relativamente al gruppo [integrazione.json](#) all'interno del quale i test utilizzano la correlazione applicativa di tipo “template”, “velocity” e “freemarker”.

Evidenze disponibili in risultati dei test del gruppo “integrazione.json”

3.2.11 Registrazione dei messaggi

I test realizzati tramite il tool [JUnit](#) verificano le funzionalità di registrazione dei contenuti dei messaggi.

I sorgenti sono disponibili in [protocolli/trasparente/testsuite/karate/src](#) relativamente ai seguenti gruppi:

- [registrazione_messaggi.dump_binario](#); viene verificata la registrazione dei messaggi inviati e ricevuta dal gateway «as is» in forma “binaria” su database.
- [registrazione_messaggi.dump_normale](#); viene verificata la precedente modalità di salvataggio in cui era il gateway ad analizzare in realtime la struttura multipart ed a salvarne i singoli elementi (envelope, attachments).

Evidenze disponibili in:

- risultati dei test del gruppo “registrazione_messaggi.dump_binario”
- risultati dei test del gruppo “registrazione_messaggi.dump_normale”

3.2.12 Header di Integrazione

I test realizzati tramite il tool **JUnit** verificano le funzionalità di integrazione con i backend per lo scambio di informazioni.

I sorgenti sono disponibili in [protocolli/trasparente/testsuite/karate/src](#) relativamente ai seguenti gruppi:

- **integrazione.autenticazione**; verifica l'integrazione che consente di generare Header HTTP utilizzabili dal backend per autenticare l'API Gateway.
- **integrazione.template**; verifica l'integrazione "template" che consente di applicare una trasformazione al messaggio.
- **integrazione.accept**; verifica l'utilizzo dell'header "Accept" per API REST.
- **integrazione.json**; verifica l'utilizzo dell'interscambio di informazioni tramite una struttura JSON.

Evidenze disponibili in:

- risultati dei test del gruppo "integrazione.autenticazione"
- risultati dei test del gruppo "integrazione.template"
- risultati dei test del gruppo "integrazione.accept"
- risultati dei test del gruppo "integrazione.json"

Sono inoltre disponibili ulteriori test realizzati tramite il tool **TestNG** i cui sorgenti sono disponibili in [protocolli/spcoop/testsuite/src](#) relativamente ai seguenti gruppi:

- **Integrazione**; vengono verificate le funzionalità di integrazione tramite header soap, header di trasporto, url e identificazione basata su contenuto, url o input.
- **IntegrazioneConnettoreHTTPCORE**; simile al precedente gruppo, viene però utilizzato il connettore "httpcore".
- **IntegrazioneConnettoreSAAJ**; simile al precedente gruppo, viene però utilizzato il connettore "saaj".
- **RichiesteApplicativeScorrette**; vengono generate richieste applicative scorrette che il gateway deve riconoscere e gestire.

Evidenze disponibili in:

- risultati dei test sugli header di integrazione

3.2.13 Encoding

I test realizzati tramite il tool **JUnit** verificano le funzionalità che impattano sull'encoding dei messaggi.

I sorgenti sono disponibili in [protocolli/trasparente/testsuite/karate/src](#) relativamente ai seguenti gruppi:

- **encoding.charset**; verifica le comunicazioni in stream, con o senza registrazione dei messaggi, la costruzione di oggetti in memoria "read only" o modificabili per trasformazioni, con charset "UTF-8", "UTF-16" e "ISO-8859-1".
- **encoding.entity_reference**; verifica le comunicazioni in stream, con o senza registrazione dei messaggi, e la costruzioni di header con richieste che contengono molti xml entity reference (2MB) che se trattate come contenuto SAAJ fanno salire il carico per il lavoro richiesto al GC (G1 Humongous Allocation).

Evidenze disponibili in:

- risultati dei test del gruppo "encoding.charset"
- risultati dei test del gruppo "encoding.entity_reference"

Sono inoltre disponibili ulteriori test realizzati tramite il tool **TestNG** che verificano la corretta gestione di messaggi che contengono caratteri non compresi nel set ASCII.

I sorgenti sono disponibili in `protocolli/spcoop/testsuite/src/.../others/XMLEncoding.java`.

Evidenze disponibili in [risultati dei test](#)

3.2.14 Plugins

I test realizzati tramite il tool **JUnit** verificano le funzionalità di plugins che consentono di personalizzare l'autenticazione, l'autorizzazione, il parsing delle risposte degli authorization server e la possibilità di registrare handlers all'interno del pipeline di gestione delle richieste e delle risposte del gateway.

I sorgenti sono disponibili in `protocolli/trasparente/testsuite/karate/src/.../plugin/test`.

Evidenze disponibili in [risultati dei test del gruppo “plugins”](#)

Sono inoltre disponibili ulteriori test realizzati tramite il tool **TestNG** che verificano la personalizzazione di handlers.

I sorgenti sono disponibili in `protocolli/spcoop/testsuite/src/.../others/Handlers.java`.

Evidenze disponibili in [risultati dei test](#)

3.2.15 Profilo «ModI»

I test realizzati tramite il tool **Karate** verificano tutte le funzionalità previsto dal profilo di interoperabilità “ModI”.

I sorgenti sono disponibili in `protocolli/modi/testsuite/src` relativamente ai seguenti gruppi:

- `org.openspcoop2.core.protocolli.modipa.testsuite.rest.bloccante`; vengono verificati i profili di interazione bloccanti e CRUD su API REST.
- `org.openspcoop2.core.protocolli.modipa.testsuite.soap.bloccante`; viene verificato il profilo di interazione bloccante su API SOAP.
- `org.openspcoop2.core.protocolli.modipa.testsuite.rest.non_bloccante.push`; viene verificato il profilo di interazione non bloccante “Push” su API REST.
- `org.openspcoop2.core.protocolli.modipa.testsuite.soap.non_bloccante.push`; viene verificato il profilo di interazione non bloccante “Push” su API SOAP.
- `org.openspcoop2.core.protocolli.modipa.testsuite.rest.non_bloccante.pull`; viene verificato il profilo di interazione non bloccante “Pull” su API REST.
- `org.openspcoop2.core.protocolli.modipa.testsuite.soap.non_bloccante.pull`; viene verificato il profilo di interazione non bloccante “Pull” su API SOAP.
- `org.openspcoop2.core.protocolli.modipa.testsuite.rest.sicurezza_messaggio`; vengono verificati tutti i pattern di sicurezza previsti dalla Linee Guida di Interoperabilità ModI per quanto concerne API REST, come descritto nella sezione *Pattern di Sicurezza ModI*.
- `org.openspcoop2.core.protocolli.modipa.testsuite.soap.sicurezza_messaggio`; vengono verificati tutti i pattern di sicurezza previsti dalla Linee Guida di Interoperabilità ModI per quanto concerne API SOAP, come descritto nella sezione *Pattern di Sicurezza ModI*.

Evidenze disponibili in:

- API REST - Pattern Interazione Bloccante
- API SOAP - Pattern Interazione Bloccante
- API REST - Pattern Interazione Non Bloccante Push

- API SOAP - Pattern Interazione Non Bloccante Push
- API REST - Pattern Interazione Non Bloccante Pull
- API SOAP - Pattern Interazione Non Bloccante Pull
- API REST - Pattern Sicurezza
- API SOAP - Pattern Sicurezza

3.2.16 Profilo «SPCoop»

I test realizzati tramite il tool [TestNG](#) verificano tutte le funzionalità previste dal profilo di interoperabilità “SPCoop”.

I sorgenti sono disponibili in [protocolli/spcoop/testsuite/src](#) relativamente ai seguenti gruppi:

- [ProfiliDiCollaborazione](#); vengono verificati i quattro profili di collaborazione: oneway, sincrono, asincronoSimmetrico e asincronoAsimmetrico.
- [PortTypes](#); vengono verificati i quattro profili di collaborazione configurati tramite accordi con port types multipli.
- [FiltroDuplicatiEGov](#); viene verificata la funzionalità di filtro e-gov dei duplicati per i 4 profili di collaborazione.
- [FunzionalitaEGov](#); vengono verificate tutte le funzionalità previste per la busta e-Gov quali oltre al filtro dei duplicati: “scadenza”, “riscontri”, “consegna in ordine”.
- [BusteEGovCampiDuplicati](#); vengono generate buste SPCoop che contengono campi duplicati e viene controllato che l’errore sia correttamente segnalato e gestito.
- [BusteEGovScorrette](#); vengono generate buste SPCoop che contengono anomalie e viene controllato che l’errore sia correttamente segnalato e gestito.
- [BusteEGovConEccezioni](#); vengono generate buste SPCoop che contengono una lista di eccezioni e viene controllato che l’eccezione sia correttamente segnalata e gestita.
- [BusteEGovNamespace](#); vengono generate buste SPCoop che contengono strutture dati dove si annidano ridefinizioni di namespace.
- [ProfiliDiCollaborazioneLineeGuida11](#); vengono verificati i quattro profili di collaborazione (oneway, sincrono, asincronoSimmetrico e asincronoAsimmetrico) rispetto alle Linee Guida 1.1 della Busta e-Gov 1.1.
- [BusteEGov11LineeGuida11](#); vengono verificate le anomalie relative rispetto alle Linee Guida 1.1 della Busta e-Gov 1.1.
- [RiconoscimentoProfiloGestione](#); viene verificata la funzionalità di identificazione del corretto profilo di gestione (vecchia versione egov o nuove linee guida).
- [ErroreApplicativoCNIPA](#) e [OpenSPCoopDetail](#); viene verificato che l’elemento “detail” in un SOAP-Fault sia valorizzato rispetto all’elemento “eGov_IT_Ecc:MessaggioDiErroreApplicativo” definito dal documento CNIPA “Porta di Dominio” e all’elemento “dettaglio-eccezione” nello schema [core/src/schemi/openspcoopDetail.xsd](#).

Evidenze disponibili in:

- risultati dei test per la gestione del profilo SPCoop
- risultati dei test per la gestione del profilo SPCoop rispetto alle linee guida 1.1
- risultati dei test effettuati con buste non corrette
- risultati dei test per il riconoscimento del corretto profilo di gestione
- risultati dei test per la gestione del dettaglio “eGov_IT_Ecc:MessaggioDiErroreApplicativo” di un SOAPFault

- risultati dei test per la gestione del dettaglio “dettaglio-eccezione” di un SOAPFault

3.2.17 API di Configurazione

I test realizzati tramite il tool [Karate](#) verificano le operazioni di configurazione del registro di Govway disponibili anche tramite la console «govwayConsole».

I sorgenti sono disponibili in [tools/rs/config/server/testsuite/src](#); i test verificano le operazioni CRUD per i seguenti oggetti del registro:

- API
- Risorse per API REST
- Servizi per API SOAP
- Azioni di servizi per API SOAP
- Allegati
- Erogazioni
- Fruizioni
- Soggetti
- Applicativi
- Applicativi di tipo server
- Ruoli
- Scope
- Stato di funzionamento dell'API

Evidenze disponibili in risultati dei test sull'API di Configurazione

3.2.18 API di Monitoraggio

I test realizzati tramite il tool [Karate](#) verificano le funzionalità di monitoraggio e statistica accedibili anche tramite console «govwayMonitor».

I sorgenti sono disponibili in [tools/rs/monitor/server/testsuite/src](#); i test verificano i seguenti gruppi:

- Ricerche di Transazioni
- Reports Statistici
- Stato di funzionamento dell'API

Evidenze disponibili in risultati dei test sull'API di Monitoraggio

Third Party Dependency Analysis

Ogni libreria terza parte utilizzata da GovWay viene sottoposta a verifica di possibile presenza di vulnerabilità di sicurezza note tramite il tool *OWASP Dependency-Check*. la cui configurazione può essere consultata nel file `mvn/dependencies/pom.xml`.

Il tool è configurato per utilizzare le seguenti base dati di vulnerabilità note:

- National Vulnerability Database;
- NPM Public Advisories;
- RetireJS;
- Sonatype OSS Index;
- CISA Known Exploited Vulnerabilities Catalog.

L'analisi viene effettuata in automatico ad ogni commit sul *master* dei sorgenti del progetto, come descritto nella sezione *OWASP Dependency-Check Jenkins Plugin*.

La verifica può essere attivata anche manualmente, effettuando il checkout dei sorgenti del progetto GovWay come descritto nella sezione *OWASP Dependency-Check Maven Plugin*.

Nel caso in cui il processo di verifica, descritto nella sezione *OWASP Dependency-Check Jenkins Plugin*, rilevasse una vulnerabilità, viene avviata una gestione della vulnerabilità come descritto in *vulnerabilityManagement*.

Altrimenti, se a valle dell'analisi della vulnerabilità rilevata, si riscontrasse un falso positivo (*vulnerabilityManagement_skip_registry*), questa verrebbe registrata come tale nella configurazione del tool *OWASP Dependency-Check*, in modo che successive verifiche non ne segnalino più la presenza. Maggiori dettagli sulla modalità di registrazione dei falsi positivi nel tool *OWASP Dependency-Check* vengono forniti nella sezione *OWASP Dependency-Check Falsi Positivi*.

Nota: Per evitare che il progetto erediti possibili vulnerabilità da software terze parti non utilizzati, tutte e sole le librerie terza parte utilizzate nel progetto govway sono definite puntualmente nei file `mvn/dependencies/*/pom.xml`.

Per ognuna di tali librerie, maven è configurato per il download puntuale del solo archivio jar interessato, escludendo esplicitamente il download ricorsivo degli archivi jar indicati come dipendenze, utilizzando l'elemento “**exclusions**”, come mostrato di seguito:

```
<dependency>
  <groupId>...</groupId>
  <artifactId>...</artifactId>
  <version>...</version>
  <exclusions>
    <exclusion>
      <groupId>*</groupId>
      <artifactId>*</artifactId>
    </exclusion>
  </exclusions>
</dependency>
```

4.1 OWASP Dependency-Check Jenkins Plugin

Ad ogni commit sul [master dei sorgenti del progetto](#) viene avviata automaticamente una verifica delle librerie terza parte nell'ambiente di [Continuous Integration Jenkins di GovWay](#).

L'analisi produce un [report di dettaglio](#) sulle vulnerabilità trovate. Per ogni vulnerabilità identificata vengono forniti maggiori dettagli come la severità, il codice identificativo e la base dati dove di appartenenza (es. [Fig. 4.1](#)).

Dependency-Check Results

SEVERITY DISTRIBUTION

1

Search


File Name	Vulnerability	Severity	Weakness
<div>— snakeyaml-1.33-gov4j-1.jar</div>	<div>OSSINDEX</div> CVE-2022-41854	<div> Medium</div>	CWE-121
<div>File Path</div>	/var/lib/jenkins/.m2/repository/org/yaml/snakeyaml/1.33-gov4j-1/snakeyaml-1.33-gov4j-1.jar		
<div>SHA-1</div>	8ced6caa339ed26a94fd597851215eeccaefd415		
<div>SHA-256</div>	658be6861d5e8eda38a7dbbfe89153dbf7d38f70828384204b4f9ba847f0a40b		
<div>Description</div>	Those using Snakeyaml to parse untrusted YAML files may be vulnerable to Denial of Service attacks (DOS). If the parser is running on user supplied input, an attacker may supply content that cause s the parser to crash by stack overflow. This effect may support a denial of service attack.		

Fig. 4.1: OWASP Dependency-Check: dettaglio di una vulnerabilità

Nella [homepage](#) dell'ambiente CI Jenkins di GovWay è anche disponibile un report che visualizza il trend delle vulnerabilità rispetto ai commit effettuati nel tempo (es. [Fig. 4.2](#)).

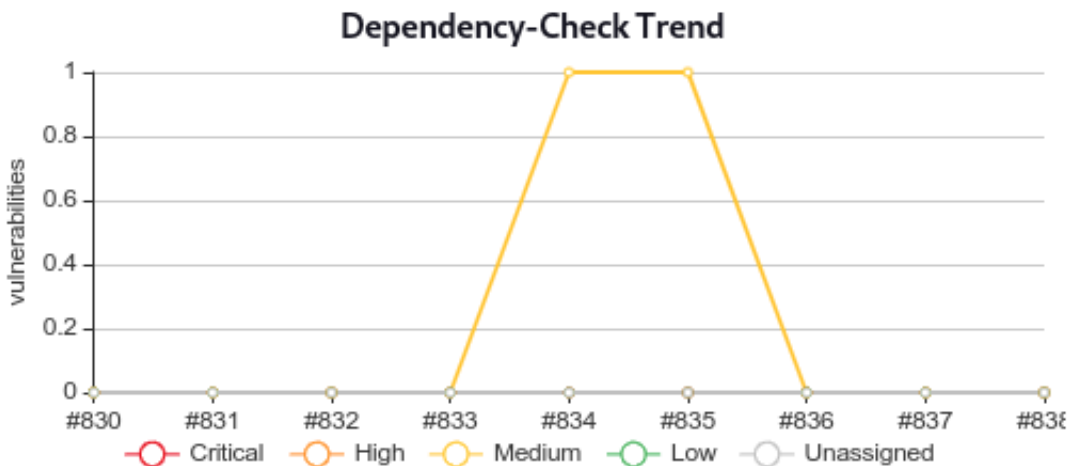


Fig. 4.2: OWASP Dependency-Check Trend

Sono inoltre disponibili report di dettaglio in vari formati ([Fig. 4.3](#)).

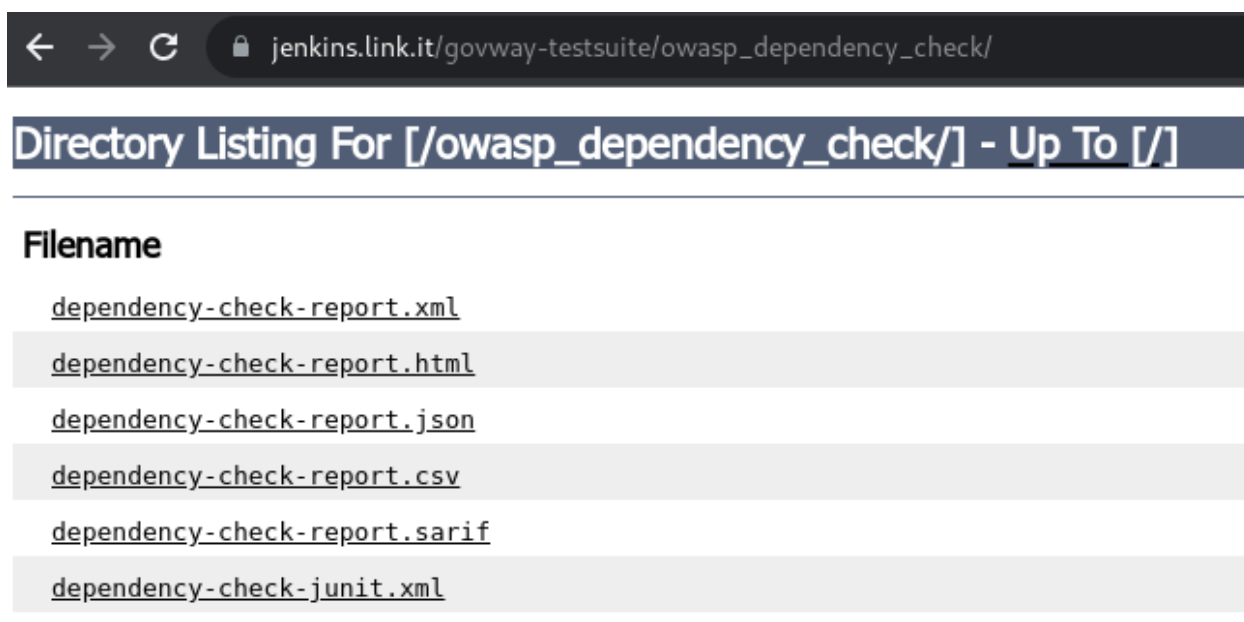


Fig. 4.3: OWASP Dependency-Check: report in vari formati

La figura [Fig. 4.4](#) mostra un esempio di report nel formato HTML.



Dependency-Check is an open source tool performing a best effort analysis of 3rd party dependencies; false positives and false negatives may exist in the analysis performed by the tool. Use of the tool and the reporting provided constitutes acceptance for use in an AS IS condition, and there are NO warranties, implied or otherwise, with regard to the analysis or its use. Any use of the tool and the reporting provided is at the user's risk. In no event shall the copyright holder or OWASP be held liable for any damages whatsoever arising out of or in connection with the use of this tool, the analysis performed, or the resulting report.

[How to read the report](#) | [Suppressing false positives](#) | [Getting Help: github issues](#)

♥ [Sponsor](#)

Project: dependencies

org.openscoop2:org.openscoop2.dependencies:1.0

Scan Information ([show all](#)):

- *dependency-check version:* 7.3.2
- *Report Generated On:* Sat, 19 Nov 2022 09:01:25 +0100
- *Dependencies Scanned:* 1779 (1610 unique)
- *Vulnerable Dependencies:* 0
- *Vulnerabilities Found:* 0
- *Vulnerabilities Suppressed:* 50
- ...

Summary

Display: [Showing Vulnerable Dependencies \(click to show all\)](#)

Dependency	Vulnerability IDs	Package	Highest Severity	CVE Count	Confidence	Evidence Count
------------	-------------------	---------	------------------	-----------	------------	----------------

Dependencies

Suppressed Vulnerabilities



This report contains data retrieved from the [National Vulnerability Database](#).

This report may contain data retrieved from the [NPM Public Advisories](#).

This report may contain data retrieved from [RetireJS](#).

This report may contain data retrieved from the [Sonatype OSS Index](#).

Fig. 4.4: OWASP Dependency-Check: html report

4.2 OWASP Dependency-Check Maven Plugin

Effettuato il checkout dei [dei sorgenti del progetto GovWay](#), è possibile avviare manualmente una analisi delle librerie terza parte utilizzando il seguente comando maven nella radice del progetto:

```
mvn verify
```

Per saltare la fase di compilazione, packaging e di test è possibile utilizzare il comando con i seguenti parametri:

```
mvn verify -Dcompile=none -Dtestsuite=none -Dpackage=none
```

Al termine dell'analisi viene prodotto un report nella directory “dependency-check-result” in differenti formati. La figura [Fig. 4.5](#) mostra un esempio di report nel formato HTML.

4.3 OWASP Dependency-Check Falsi Positivi

Librerie utilizzate solo a scopo di test

Le librerie terza parte utilizzate solamente nelle testsuite interne del prodotto, che non insistono negli archivi binari rilasciati, non vengono considerate tra i check di vulnerabilità “owasp”. Il motivo risiede nel fatto che alcune batterie di test richiedono versioni storiche di librerie, alcune delle quali possiedono anche vulnerabilità. Per evitare la verifica, le librerie utilizzate solamente dalle testsuite e non dai componenti runtime sono marcate con lo scope “test” come mostrato nell'esempio seguente:

```
<dependency>
  <groupId>org.apache.axis</groupId>
  <artifactId>axis</artifactId>
  <version>1.4</version>
  <exclusions>
    <exclusion>
      <groupId>*</groupId>
      <artifactId>*</artifactId>
    </exclusion>
  </exclusions>
  <scope>test</scope>
</dependency>
```

Falsi Positivi

Nell'utilizzo del plugin vengono aggiunte le configurazioni che permettono di registrare dei falsi positivi rispetto al progetto, individuati nella vulnerabilityManagement. Di seguito il frammento del file [mvn/dependencies/pom.xml](#) che evidenzia come venga utilizzato il plugin owasp configurato con i suppressionFiles:

```
<plugin>
  <groupId>org.owasp</groupId>
  <artifactId>dependency-check-maven</artifactId>
  <version>7.3.2</version>
  <executions>
    <execution>
      <id>check owasp</id>
      <phase>verify</phase>
      <configuration>
        <autoUpdate>true</autoUpdate>
        <failBuildOnAnyVulnerability>false</failBuildOnAnyVulnerability>
        <outputDirectory>../../dependency-check-result</outputDirectory>
      </configuration>
    </execution>
  </executions>
</plugin>
```

(continues on next page)



Dependency-Check is an open source tool performing a best effort analysis of 3rd party dependencies; false positives and false negatives may exist in the analysis performed by the tool. Use of the tool and the reporting provided constitutes acceptance for use in an AS IS condition, and there are NO warranties, implied or otherwise, with regard to the analysis or its use. Any use of the tool and the reporting provided is at the user's risk. In no event shall the copyright holder or OWASP be held liable for any damages whatsoever arising out of or in connection with the use of this tool, the analysis performed, or the resulting report.

[How to read the report](#) | [Suppressing false positives](#) | [Getting Help: github issues](#)

♥ [Sponsor](#)

Project: dependencies

org.openscoop2:org.openscoop2.dependencies:1.0

Scan Information ([show all](#)):

- *dependency-check version:* 7.3.2
- *Report Generated On:* Sat, 19 Nov 2022 09:01:25 +0100
- *Dependencies Scanned:* 1779 (1610 unique)
- *Vulnerable Dependencies:* 0
- *Vulnerabilities Found:* 0
- *Vulnerabilities Suppressed:* 50
- ...

Summary

Display: [Showing Vulnerable Dependencies \(click to show all\)](#)

Dependency	Vulnerability IDs	Package	Highest Severity	CVE Count	Confidence	Evidence Count
------------	-------------------	---------	------------------	-----------	------------	----------------

Dependencies

Suppressed Vulnerabilities



This report contains data retrieved from the [National Vulnerability Database](#).
This report may contain data retrieved from the [NPM Public Advisories](#).
This report may contain data retrieved from [RetireJS](#).
This report may contain data retrieved from the [Sonatype OSS Index](#).

Fig. 4.5: OWASP Dependency-Check: html report

(continua dalla pagina precedente)

```
        <format>ALL</format>
        <suppressionFiles>
            <suppressionFile>${owasp.falsePositives.dir}/swagger-codegen-
↪linkit.xml</suppressionFile>
            <suppressionFile>${owasp.falsePositives.dir}/console-back-
↪office.xml</suppressionFile>
            ...
        </suppressionFiles>
    </configuration>
    <goals>
        <goal>aggregate</goal>
    </goals>
</execution>
</executions>
</plugin>
```

Esaminando nel dettaglio i file che definiscono i falsi positivi:

- swagger-codegen-linkit.xml: esclude i jar inclusi nel file [mvn/dependencies/swagger-codegen/pom.xml](#) poichè vengono utilizzati solamente durante lo sviluppo per generare alcune classi e non a runtime dal Gateway.
- commons-discovery.xml: vulnerabilityManagement_skip_registry_CVE-2022-0869
- snakeyaml.xml: vulnerabilityManagement_skip_registry_CVE-2022-38752
- spring-web.xml: vulnerabilityManagement_skip_registry_CVE-2016-1000027
- spring-security-crypto.xml: vulnerabilityManagement_skip_registry_CVE-2020-5408
- xercesImpl.xml: vulnerabilityManagement_skip_registry_CVE-2017-10355
- console-back-office.xml: esclude i jar inclusi nel file [mvn/dependencies/faces/pom.xml](#) poichè utilizzati dalle console di gestione e monitoraggio adibite a funzioni di backoffice che non devono essere esposte al pubblico.

Nota: È in corso una attività di revisione dei jar utilizzati dalle console al fine di superare tutte le vulnerabilità note.
