

---

# Processo di Validazione

*Release 3.3.10*

**Link.it**

**20 gen 2023**



<b>1</b>	<b>Introduzione</b>	<b>1</b>
<b>2</b>	<b>Static Code Analysis</b>	<b>3</b>
2.1	SpotBugs Warnings Jenkins Plugin . . . . .	4
2.2	SpotBugs Eclipse Plugin . . . . .	4
2.3	SpotBugs Maven Plugin . . . . .	7
<b>3</b>	<b>Dynamic Analysis</b>	<b>13</b>
3.1	Functional tests . . . . .	13
3.2	Security Tests . . . . .	17
<b>4</b>	<b>Third Party Dependency Analysis</b>	<b>25</b>
4.1	OWASP Dependency-Check Jenkins Plugin . . . . .	26
4.2	OWASP Dependency-Check Maven Plugin . . . . .	29
4.3	OWASP Dependency-Check Falsi Positivi . . . . .	29



---

## Introduzione

---

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 il tool [SpotBugs](#).
- *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 tramite il tool [SpotBugs](#), cercando pattern riconducibili a bug improbabili da individuare tramite test dinamici (*Dynamic Analysis*).

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

Effettuato il checkout dei sorgenti del progetto GovWay, è possibile anche avviare manualmente una analisi statica come descritto nella sezione *SpotBugs Maven Plugin*.

In ogni caso, 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*.

---

**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.

---

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.

## 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. 2.1](#)).

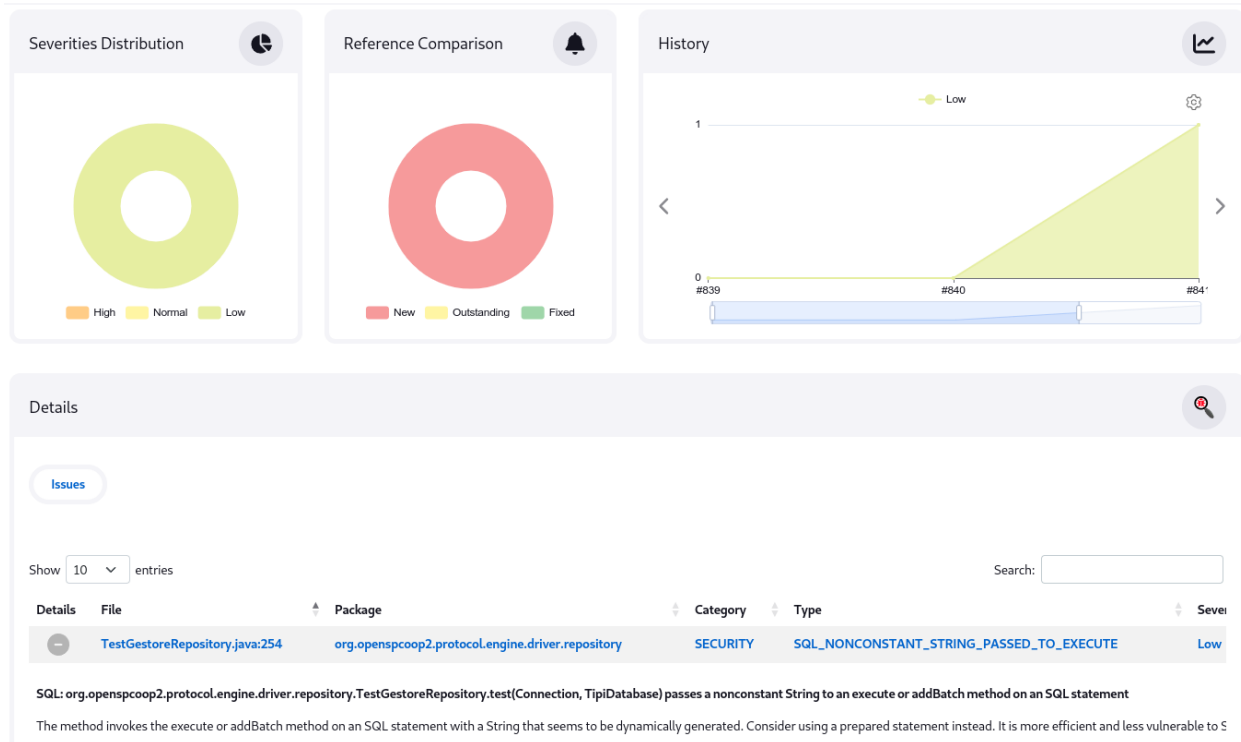


Fig. 2.1: SpotBugs: dettaglio di una vulnerabilità

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. 2.2](#)).

Sono inoltre disponibili [report di dettaglio in vari formati](#) ([Fig. 2.3](#)).

La figura [Fig. 2.4](#) mostra un esempio di report nel formato HTML.

## 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. 2.5](#).

Impostare i seguenti criteri di analisi statica accedendo alla sezione «Window -> Preferences -> Java -> SpotBugs», come mostrato nella figura [Fig. 2.6](#):

- un livello "Maximal" per il parametro "Analysis Effort";
- un livello "Low" per il parametro "Minimum confidence to report";



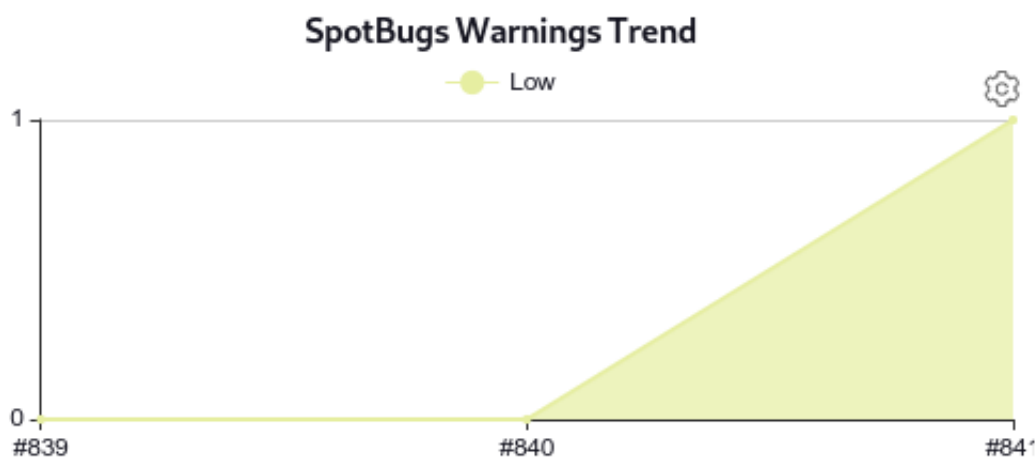


Fig. 2.2: SpotBugs Warnings Trend

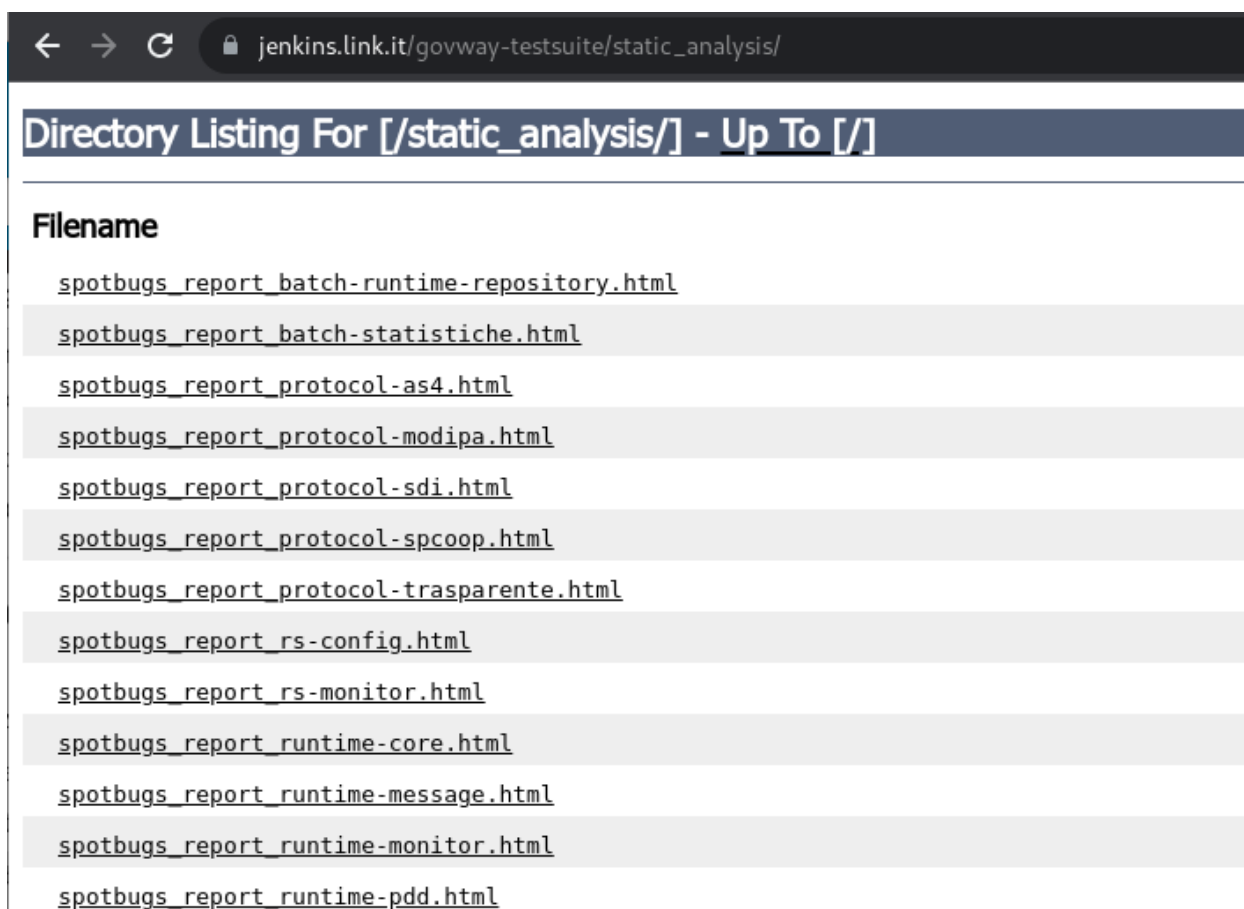


Fig. 2.3: SpotBugs: report in vari formati

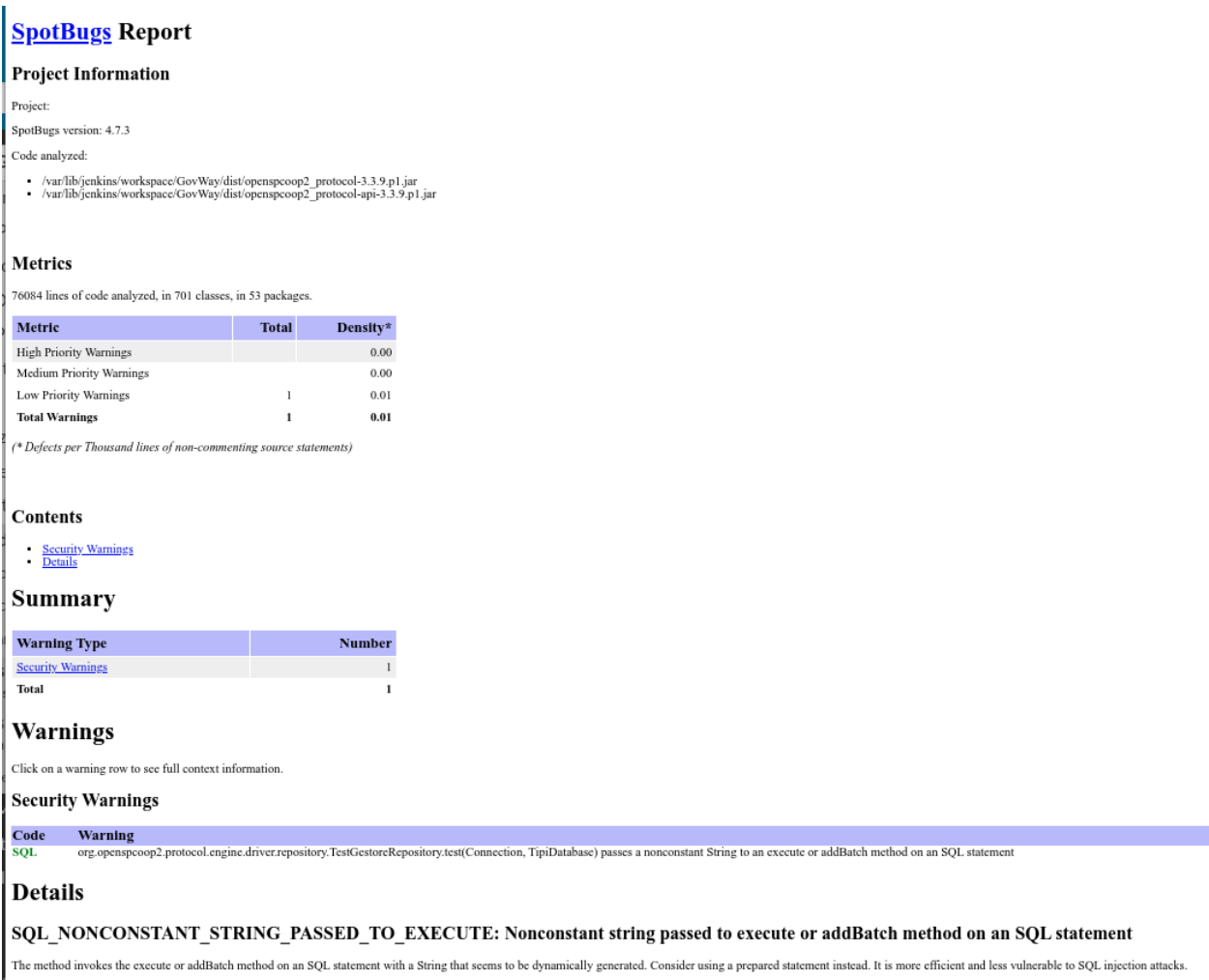


Fig. 2.4: SpotBugs: html report

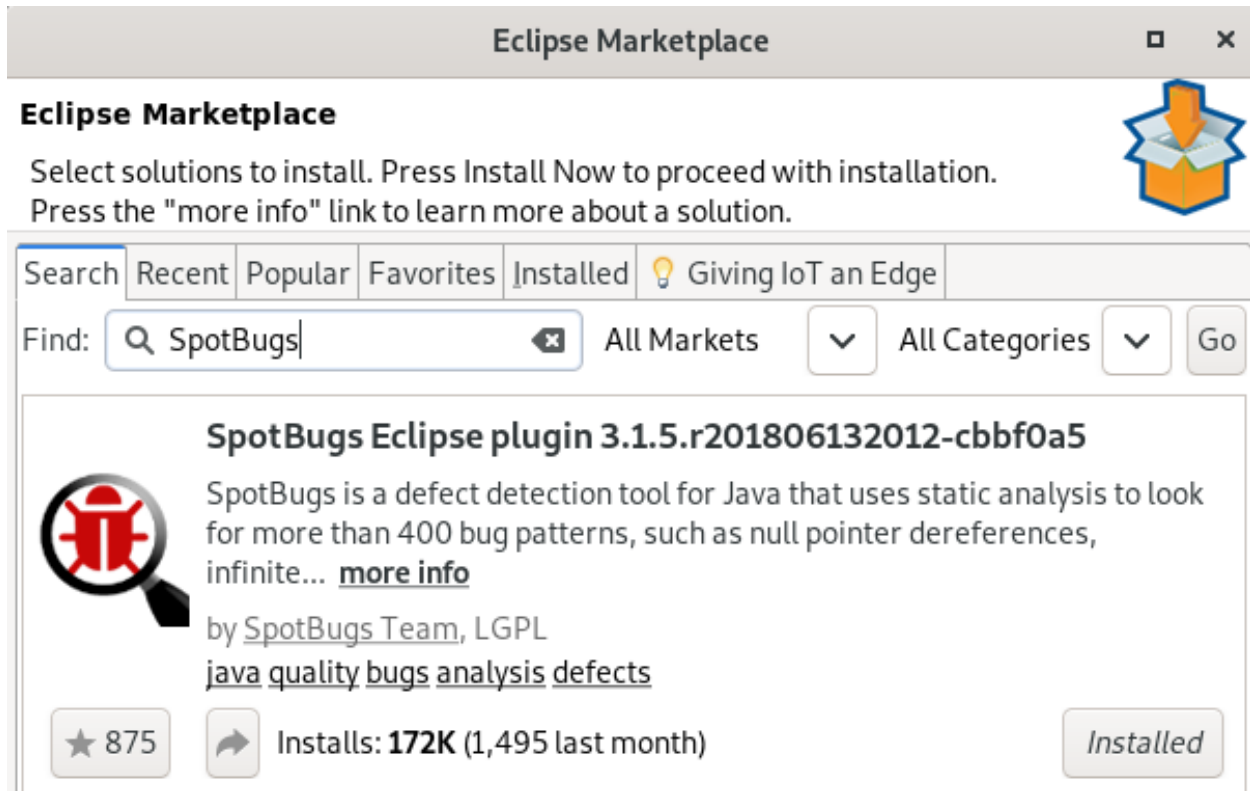


Fig. 2.5: SpotBugs Eclipse Plugin: marketplace

- un “rank” impostato al valore “15”.

Deve inoltre essere caricato il filtro che esclude alcuni **falsi positivi** come mostrato nella figura Fig. 2.7.

L’analisi statica dei sorgenti è adesso effettuabile selezionando il progetto “op2\_3.3.dev” con il tasto destro e cliccando sulla voce “SpotBugs -> Find Bugs” come mostrato nella figura Fig. 2.8.

Eventuali bug individuati vengono evidenziati sulla singola classe come ad esempio viene mostrato nella figura Fig. 2.9.

## 2.3 SpotBugs 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 -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 package analizzato.

Gli identificativi dei package 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;

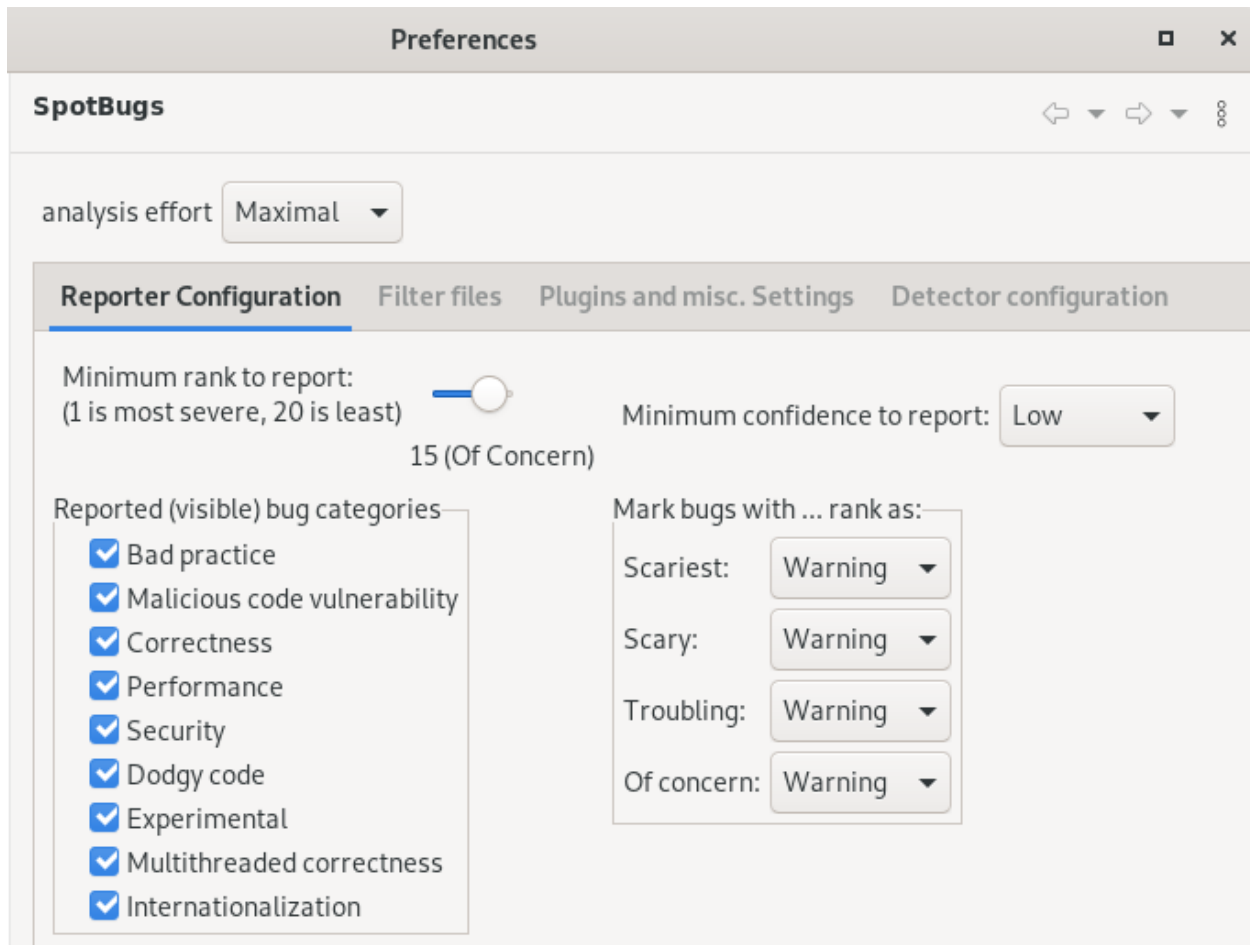


Fig. 2.6: SpotBugs Eclipse Plugin: configurazione

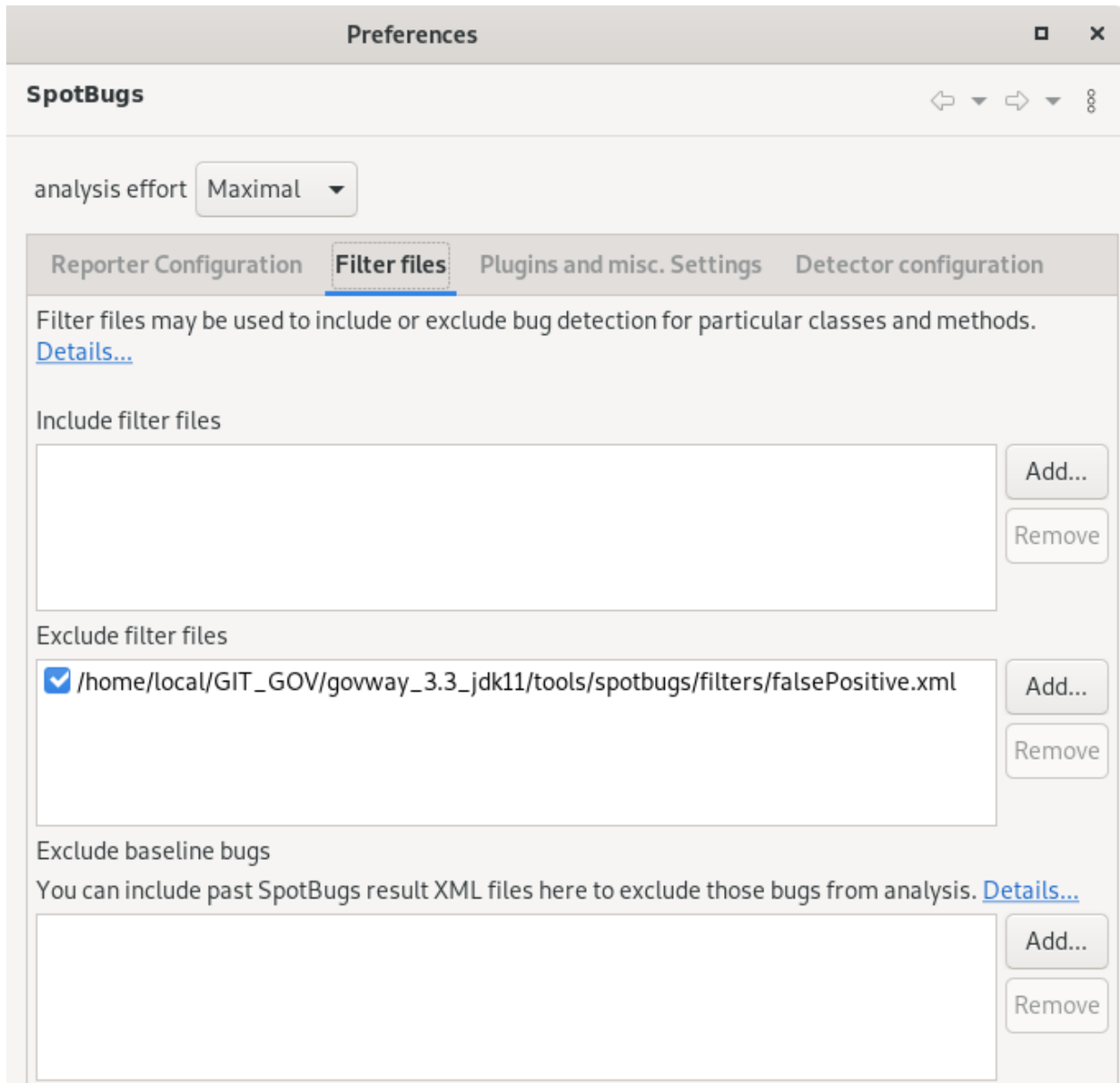


Fig. 2.7: SpotBugs Eclipse Plugin: filtro

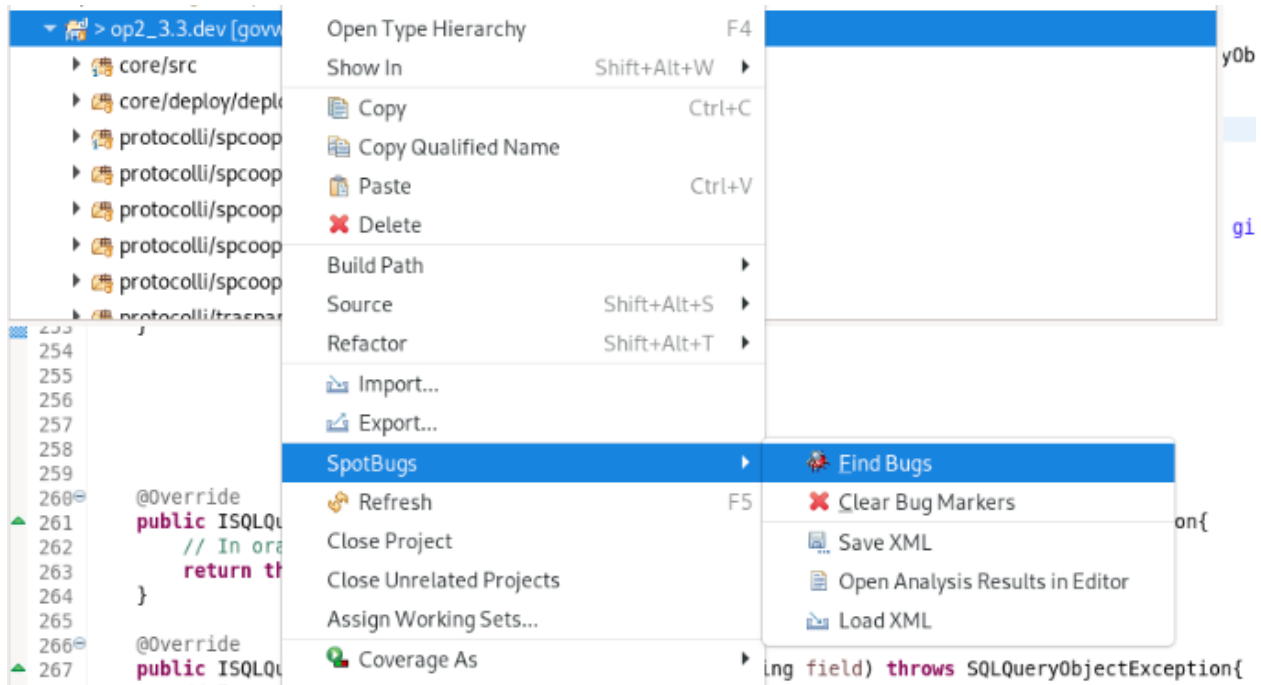


Fig. 2.8: SpotBugs Eclipse Plugin: find bugs

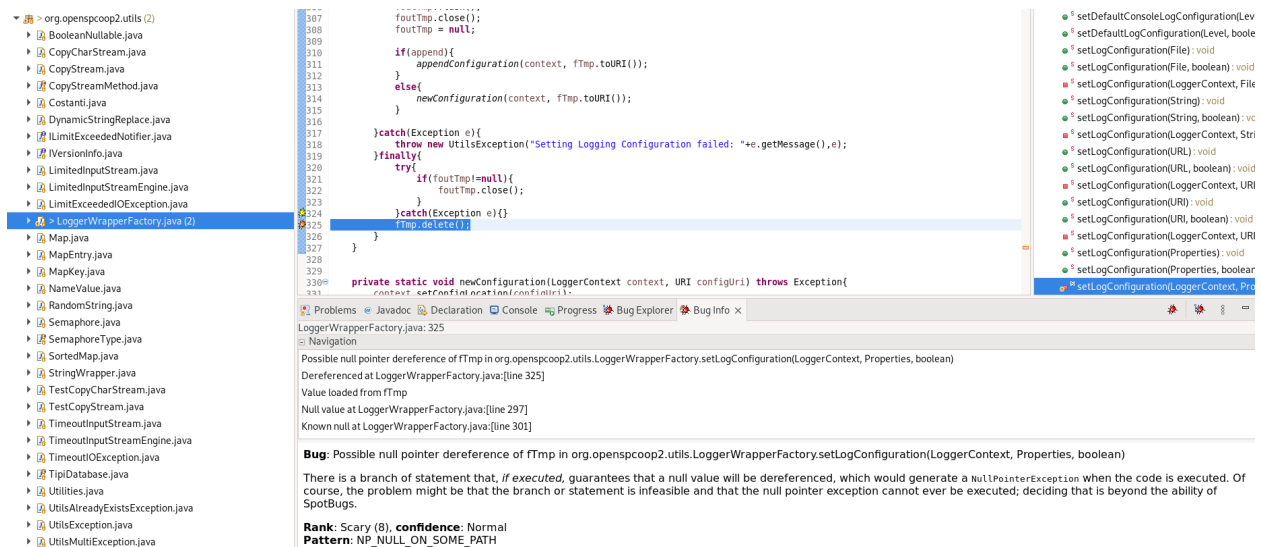


Fig. 2.9: SpotBugs Eclipse Plugin: esempio di bug

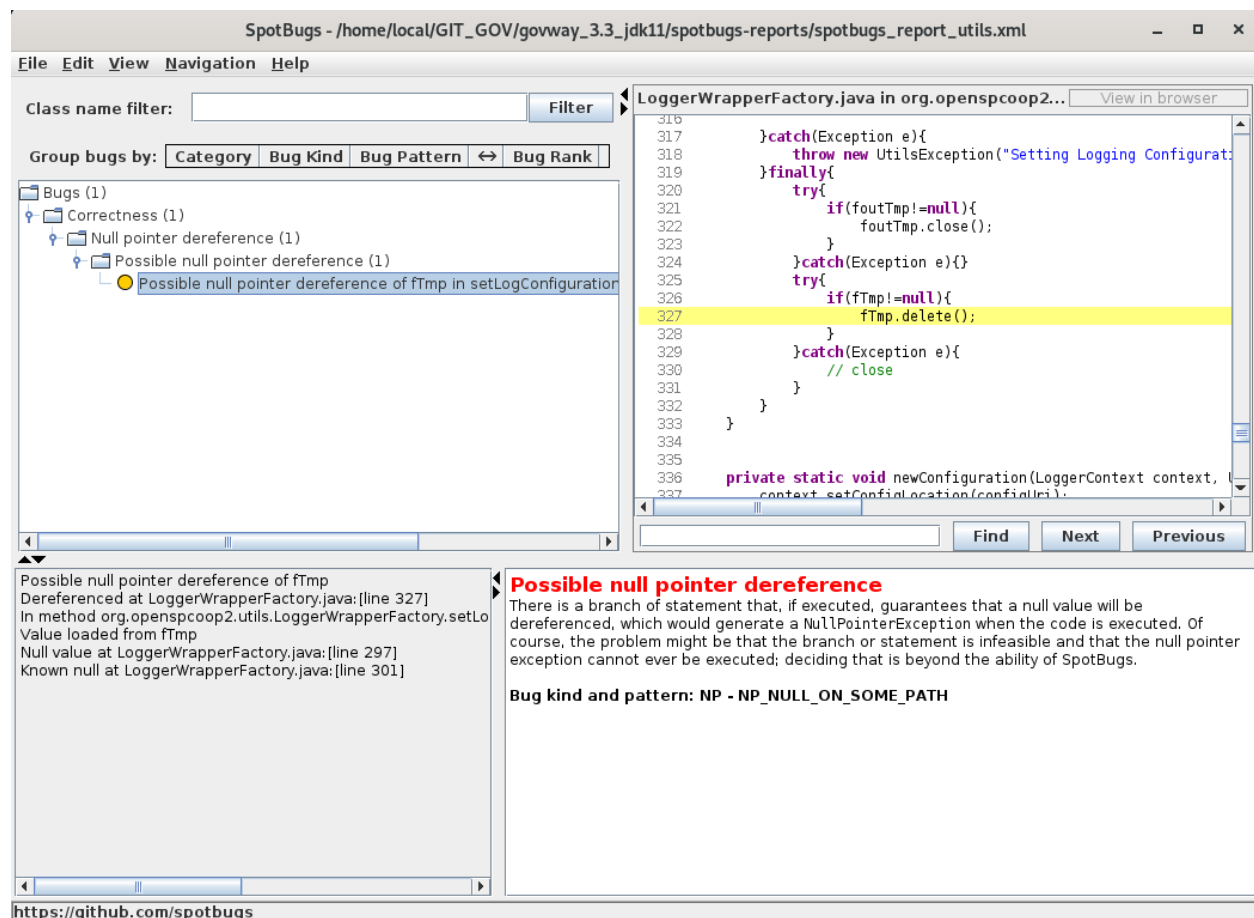
- profili di interoperabilità: protocol-as4, protocol-modipa, protocol-sdi, protocol-spcoop, protocol-trasparente;
- console web: web-lib-audit, web-lib-mvc, web-lib-queue, web-lib-users, web-loader, web-govwayConsole, web-govwayMonitor;
- api di configurazione e monitoraggio: rs-config, rs-monitor;
- batch: batch-statistiche, batch-runtime-repository.

Per evitare la verifica di alcuni package 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
```

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. 2.6 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 -Dwaspp=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.



---

## 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](#). Maggiori dettagli vengono forniti nella sezione *Functional tests*.

All'interno degli oltre 13.000 test precedentemente indicati, 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 e le console di configurazione e monitoraggio come descritto nella sezione *OWASP Zed Attack Proxy (ZAP)*.

### 3.1 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. 3.1: TestNG: dettagli dei test

Test Result



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. 3.2: 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. 3.1 e Fig. 3.2).

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. 3.3 e Fig. 3.4).

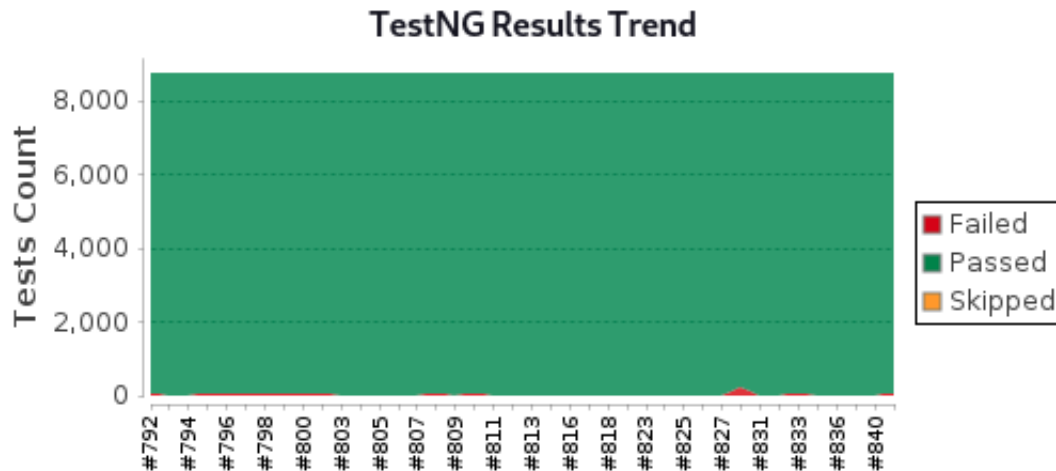


Fig. 3.3: TestNG Results Trend

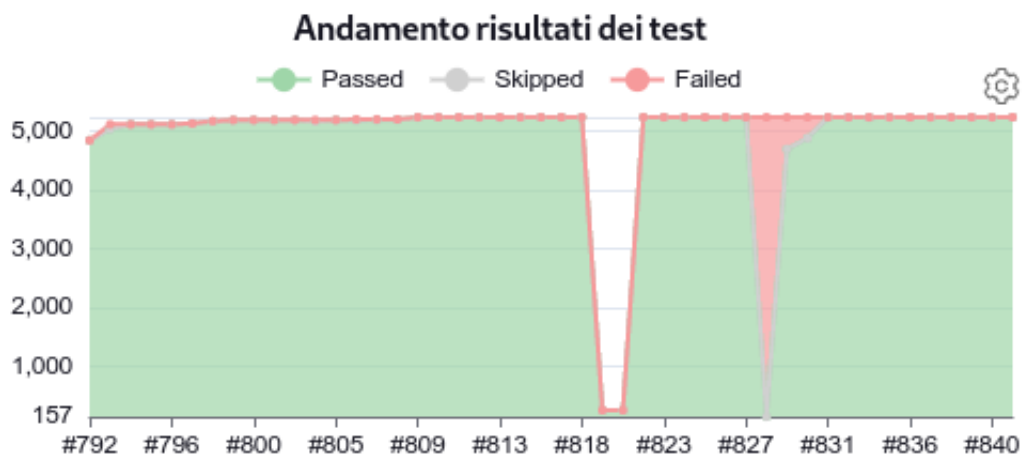


Fig. 3.4: JUnit Results Trend

Sono inoltre disponibili report di dettaglio in vari formati (Fig. 3.5).

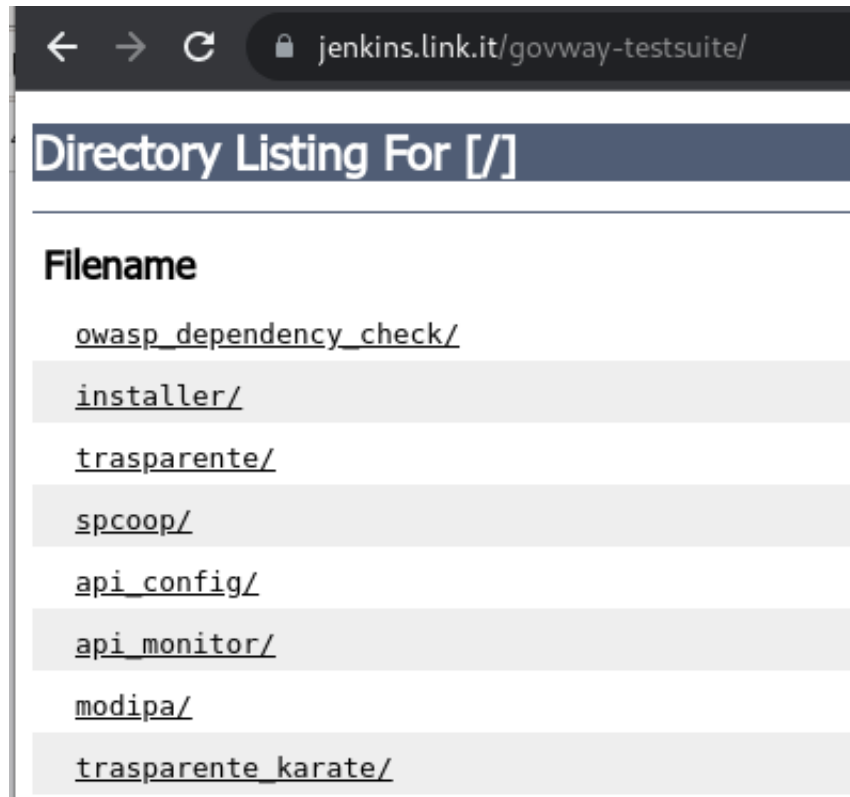


Fig. 3.5: Tests: report in vari formati

## 3.2 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.

### 3.2.1 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.openscoop2.protocol.trasparente.testsuite.units.soap.authn`
- `org.openscoop2.protocol.trasparente.testsuite.units.rest.auth`

Evidenze disponibili in:

- Autenticazione per le API Fruite
- Autenticazione per le API Erogate

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 Erogate.

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”

### 3.2.2 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 Erogate

Altri test vengono realizzati tramite il tool JUnit e 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`
- `org.openspcoop2.core.protocolli.trasparente.testsuite.token.negoziatore`
- `org.openspcoop2.core.protocolli.trasparente.testsuite.token.attribute_authority`

Evidenze disponibili in:

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

### 3.2.3 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](#)
- [org.openspcoop2.core.protocolli.modipa.testsuite.soap.sicurezza\\_messaggio](#)

Evidenze disponibili in:

- [API REST](#)
- [API SOAP](#)

### 3.2.4 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](#)

### 3.2.5 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/utills/src/org/openspcoop2/utills/test](#) relativamente ai seguenti package:

- [org.openspcoop2.utills.test.security](#)
- [org.openspcoop2.utills.test.certificate](#)
- [org.openspcoop2.utills.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

### 3.2.6 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/utls/src/org/openspcoop2/utls/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

### 3.2.7 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

### 3.2.8 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



### 3.2.9 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. 3.6](#)).

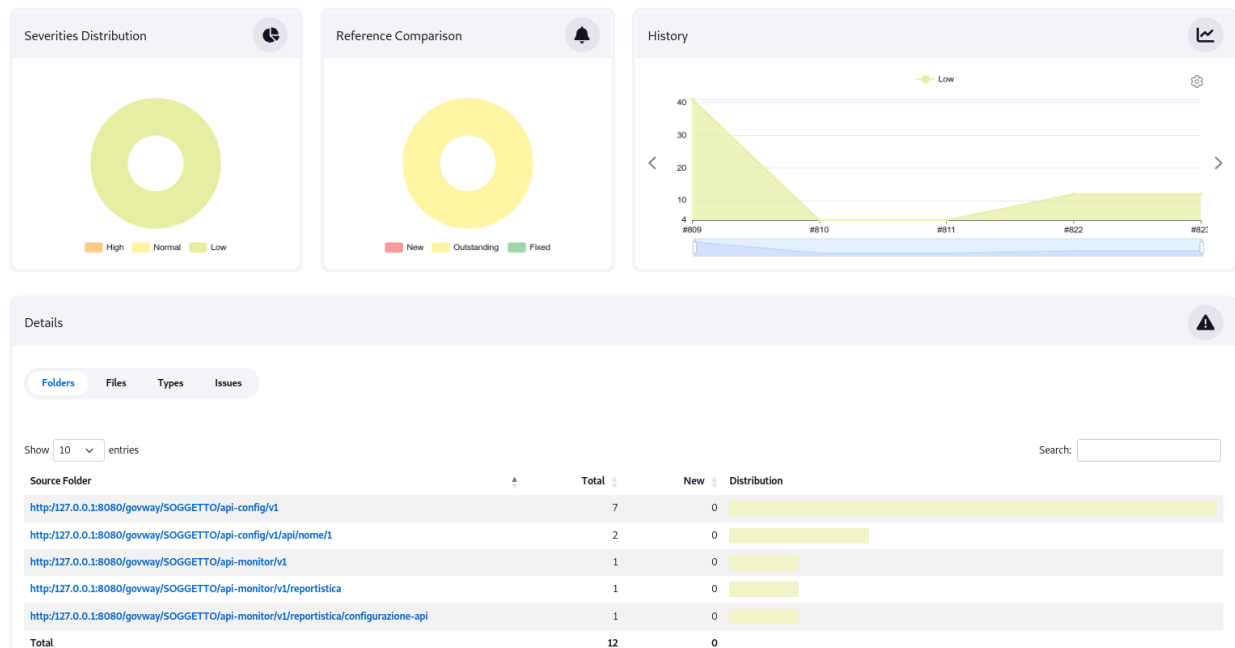


Fig. 3.6: OWASP ZAP: dettaglio di una vulnerabilità

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. 3.7](#)).

Sono inoltre disponibili [report di dettaglio in vari formati](#) ([Fig. 3.8](#)).

La figura [Fig. 3.9](#) mostra un esempio di report nel formato HTML.

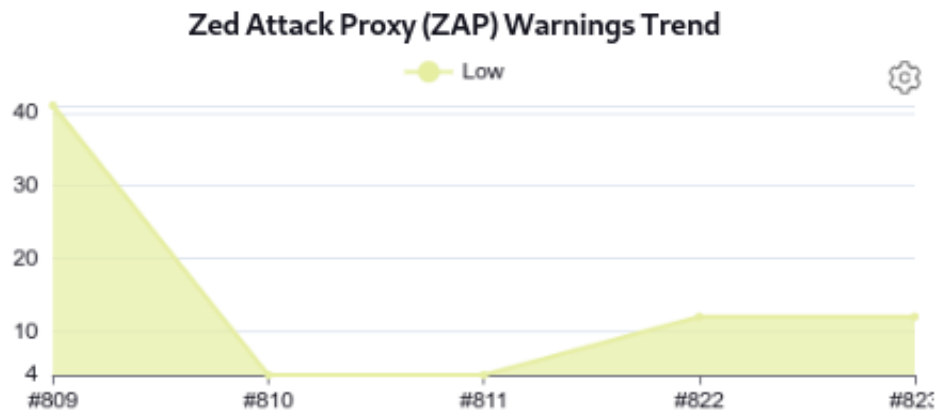



Fig. 3.7: OWASP ZAP Warnings Trend



Fig. 3.8: OWASP ZAP: report in vari formati


**GovWay API di configurazione**

Analisi delle API di GovWay per la configurazione

**Site:** <http://127.0.0.1:8080/govway/SOGGETTO/api-config/v1>

**Generated on** Thu, 1 Dec 2022 10:09:29

**Summary of Alerts**

Risk Level	Number of Alerts
High	0
Medium	0
Low	3
Informational	0
False Positives:	0

**Alerts**

Name	Risk Level	Number of Instances
<a href="#">Application Error Disclosure</a>	Low	1
<a href="#">Information Disclosure - Debug Error Messages</a>	Low	1
<a href="#">X-Content-Type-Options Header Missing</a>	Low	7

**Alert Detail**

Low	Application Error Disclosure
Description	This page contains an error/warning message that may disclose sensitive information like the location of the file that produced the unhandled exception. This information can be used to launch further attacks against the web application. The alert could be a false positive if the error message is found inside a documentation page.
URL	<a href="http://127.0.0.1:8080/govway/SOGGETTO/api-config/v1/api/home/1/descrizione?profilo=APIGateway&amp;soggetto=ENTE">http://127.0.0.1:8080/govway/SOGGETTO/api-config/v1/api/home/1/descrizione?profilo=APIGateway&amp;soggetto=ENTE</a>
Method	PUT
Parameter	
Attack	
Evidence	HTTP/1.1 500
Instances	1
Solution	Review the source code of this page. Implement custom error pages. Consider implementing a mechanism to provide a unique error reference/identifier to the client (browser) while logging the details on the server side and not exposing them to the user.
Reference	
CWE Id	<a href="#">200</a>
WASC Id	13
Plugin Id	<a href="#">90022</a>

Fig. 3.9: 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
```

---

### 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 CVE-2022-41854</div>	<div><div></div>Medium</div>	CWE-121
File Path	/var/lib/jenkins/.m2/repository/org/yaml/snakeyaml/1.33-gov4j-1/snakeyaml-1.33-gov4j-1.jar		
SHA-1	8ced6caa339ed26a94fd597851215eeccaefd415		
SHA-256	658be6861d5e8eda38a7dbbfe89153dbf7d38f70828384204b4f9ba847f0a40b		
Description	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](#)).

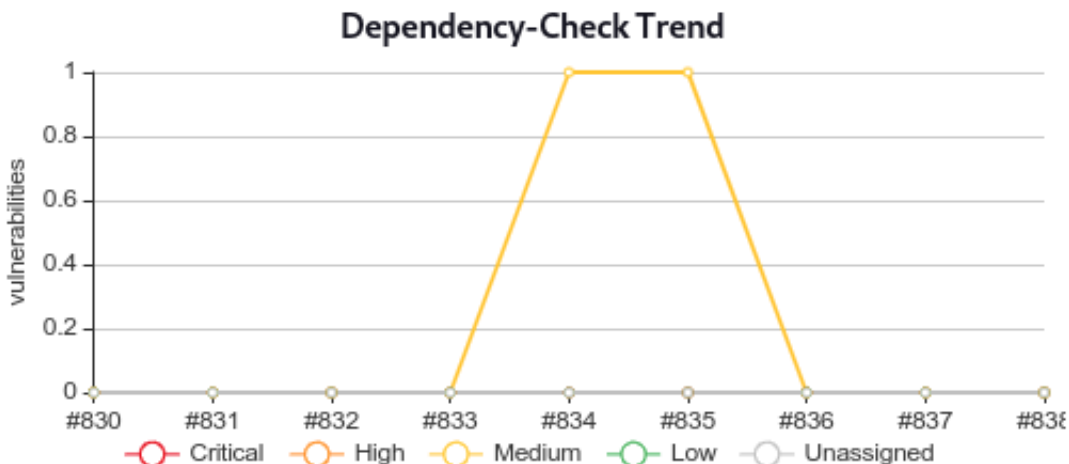


Fig. 4.2: OWASP Dependency-Check Trend

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

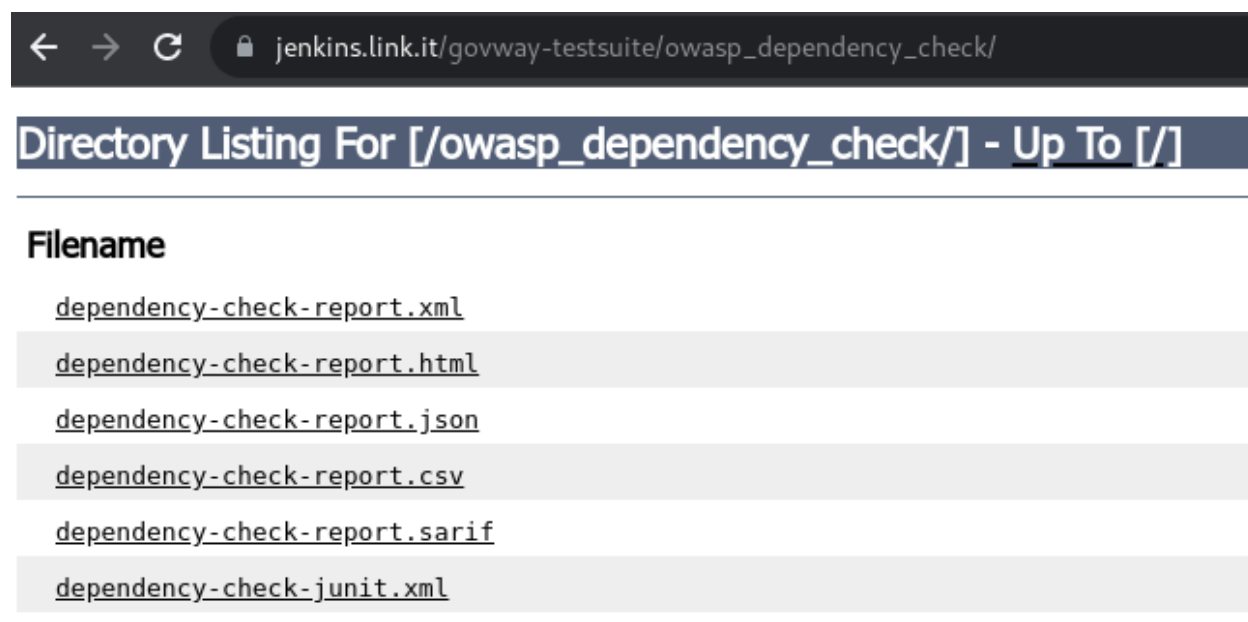


Fig. 4.3: OWASP Dependency-Check: report in vari formati

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 test suite 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 test suite 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.

---