POLITECNICO DI TORINO

Corso di Laurea Magistrale in Ingegneria Informatica

Tesi di Laurea Magistrale

Valutazione di Strumenti di Analisi Statica della Sicurezza su Applicazioni Distribuite Open Source

Relatore	Candidato
Prof. Riccardo SISTO	Giuseppina IMPAGNATIELLO

Anno Accademico 2021/2022

Sommario

L'obiettivo di questo lavoro di tesi è quello di valutare alcuni strumenti informatici utili all'analisi statica della sicurezza del codice, e di confrontarne le prestazioni in una scala di misurazione comune.

L'analisi statica della sicurezza (anche detta SAST, Static Application Security Testing) è una metodologia di analisi utilizzata per ispezionare il codice sorgente, al fine di individuare delle eventuali vulnerabilità senza la necessità di eseguire effettivamente il programma. Essa sta ormai divenendo parte integrante dello sviluppo di applicazioni web distribuite, e parallelamente il mercato degli strumenti automatizzati volti ad effettuare questo tipo di analisi è in continuo sviluppo.

Nel corso di questo lavoro di tesi il primo obiettivo è stato dunque quello di selezionare alcuni di questi strumenti in base a dei criteri predeterminati, e contemporaneamente di individuare alcune applicazioni Open Source distribuite che potessero costituire un campione vario e omogeneo. Il passo successivo è stato quello di eseguire le analisi in modo automatizzato, per poi effettuare una revisione manuale dei risultati necessaria a confermare o a smentire le segnalazioni restituite dagli strumenti. Infine l'ultimo obiettivo è stato quello di misurare alcuni parametri relativi alle prestazioni, al fine di fornire una valutazione e un confronto diretto tra gli strumenti. E' stata adottata a questo scopo la metodologia di scoring proposta dall'OWASP Foundation, la quale permette di assegnare un punteggio quantitativo alle prestazioni dei vari strumenti, che tenga globalmente conto di più indicatori: ciò ha portato a poter stabilire quali fossero gli strumenti più accurati sia a livello complessivo, sia scendendo nello specifico delle tipologie e delle categorie di vulnerabilità.

I risultati ottenuti sono generalmente in linea con quanto ci si aspettava, e potrebbero essere ulteriormente approfonditi mediante l'uso di strumenti di natura avanzata o di un campione più vario. Allo stesso modo, si potrebbero condurre ulteriori tipologie di ricerca che integrino i risultati già ottenuti, ad esempio mediante l'inserimento deliberato di specifiche vulnerabilità all'interno del codice.

Ringraziamenti

Grazie alla mia famiglia, che mi ha sostenuta in tutto il mio percorso universitario, spingendomi sempre a inseguire i miei sogni.
Grazie ai miei amici, vicini e lontani, che hanno reso questi anni unici e spensierati.
Grazie a chiunque abbia creduto nelle mie potenzialità e mi abbia guidata e supportata nella mia crescita accademica e personale.

"È dall'ironia che comincia la libertà."
Victor Hugo

Indice

Elenco delle tabelle

Elenco delle figure

Acronimi

CI/CD
Continuous Integration/ Continuous Delivery

CWE
Common Weakness Enumeration

DAST
Dynamic Application Security Testing

IDE
Integrated Development Environment

LOC
Lines Of Code

NIST
National Institute of Standards and Technologies

OWASP
Open Web Application Security Project

SANS
SysAdmin Audit Networking and Security

SAST
Static Application Security Testing

SDLC
Software Development Life Cycle

Capitolo 1

Introduzione

In questo primo Capitolo verrà introdotto il lavoro di tesi, descrivendo i concetti che sono alla base dell'Analisi Statica della sicurezza, i vantaggi e gli svantaggi di questa tipologia di analisi e le principali tecniche utilizzate. Verrà poi descritto lo stato dell'arte nell'ambito della valutazione degli strumenti automatizzati, evidenziando il problema da risolvere e l'approccio metodologico che si è deciso di adottare.

1.1 Introduzione alla SAST Analysis

Secondo il Forrester Report del 2021 sullo stato della sicurezza delle applicazioni [1], commissionato dalla Palo Alto Networks, nel 2020 due attacchi informatici su tre provenienti da fonti esterne sono stati condotti attraverso applicazioni web (39% dei casi) o sfruttando vulnerabilità del software (30% dei casi).
Risulta dunque fondamentale cercare di individuare tempestivamente le potenziali vulnerabilità di sicurezza presenti nel codice, includendo nel ciclo di vita del software (SDLC, Software Development Life Cycle) varie fasi di analisi, diverse nella metodologia e nello scopo a seconda del particolare stadio di sviluppo. Questo lavoro di tesi si concentra su un particolare tipo di analisi: *l'analisi statica della sicurezza* (in inglese SAST, Static Application Security Testing), uno dei principali approcci utilizzati dagli sviluppatori per individuare potenziali vulnerabilità di sicurezza e di qualità.

Vantaggi della SAST Analysis

La SAST Analysis è caratterizzata dal fatto di essere eseguita senza lanciare il programma, a differenza della DAST (Dynamic Application Security Testing), che invece analizza l'effettivo comportamento del codice a run time. L'assenza di una necessità di esecuzione presenta l'innegabile beneficio di poter condurre i test in

una fase molto precoce del ciclo di vita dello sviluppo, e di poter dunque traslare indietro nel tempo la scoperta di eventuali vulnerabilità. Ciò rappresenta un vantaggio significativo in termini sia di sviluppo sicuro delle applicazioni che di costi di correzione: secondo il NIST (National Institute of Standards and Technologies), correggere una vulnerabilità una volta che il codice è in produzione può costare fino a 6 volte rispetto a una vulnerabilità scoperta durante la fase di programmazione [2].

La SAST Analysis viene principalmente condotta in white-box, ossia in contesti in cui il codice sorgente è noto, tanto da essere un valido supporto alla code review (analisi di revisione del codice effettuata manualmente dagli sviluppatori). Tuttavia ci sono occasioni in cui i test vengono condotti in black-box: è il caso di codici in cui siano presenti librerie di terze parti, o di contesti in cui, pur essendo disponibile il codice sorgente, è preferibile analizzare il codice oggetto poiché meno soggetto ad ambiguità nell'interpretazione.

L'analisi statica della sicurezza viene eseguita su tutto il codice a disposizione, permettendo di individuare delle vulnerabilità presenti in eventuali sezioni che vengono eseguite solo in condizioni molto rare, e che dunque potrebbero non essere individuate con strumenti di analisi dinamica; inoltre, viene mostrata allo sviluppatore l'esatta locazione della vulnerabilità, permettendone una correzione rapida ed efficace. L'utilizzo di strumenti automatizzati garantisce infine una significativa riduzione dei tempi di analisi rispetto alla revisione totalmente manuale.

Svantaggi della SAST Analysis

La SAST Analysis presenta tuttavia degli svantaggi, che ne rendono necessaria l'integrazione con strumenti di analisi dinamica, al fine di avere un quadro più completo e puntuale delle varie vulnerabilità presenti in un determinato codice.
Un primo evidente svantaggio è la mancanza di contesto: l'analisi statica prende infatti in considerazione il comportamento del programma a fronte di tutti i possibili input, non tenendo conto delle intenzioni del programmatore. In generale, si può affermare che la SAST Analysis sia piuttosto soggetta all'individuazione di falsi positivi, ossia al riportare vulnerabilità che, a una revisione umana, si rivelano essere innocue. In questo senso si deve intendere l'analisi statica come un'attività non totalmente automatizzabile, poiché si rendono necessari controlli manuali di finalizzazione.

Tecniche di Analisi

La SAST Analysis comprende al suo interno un ventaglio di più tecniche [3]:

- Analisi lessicale: il codice viene ripulito dai commenti e suddiviso in tokens, di cui si valuta la conformità a livello lessicale.

- Analisi strutturale: vengono esaminate le strutture specifiche del linguaggio, identificando delle vulnerabilità nel design, nella dichiarazione o nell'uso di variabili e funzioni, e verificando che non ci siano inconsistenze con le pratiche di programmazione sicura.

- Control-Flow Analysis: controlla l'ordine delle operazioni eseguite, permettendo ad esempio di individuare vulnerabilità nella trasmissione dei cookies, configurazioni errate prima dell'utilizzo di una variabile, variabili non inizializzate.

- Data-Flow Analysis: individua dei punti nel codice in cui potrebbero essere immessi dati non validati, e ne segue il flusso, determinando se una sanitizzazione avvenga prima dell'uso o meno.

- Controllo delle logiche temporali: con dei model checkers si controlla che il modello ricavato dal codice soddisfi determinate proprietà (in caso contrario, viene fornito un contro-esempio), mentre coi theorem provers si punta a individuare una prova che una certa proprietà, stabilita come assioma, sia vera in ogni interpretazione del sistema formale.

- Esecuzione simbolica: è in realtà un tipo di analisi ibrida tra l'analisi statica e quella dinamica, poiché il codice viene effettivamente eseguito, ma prendendo come input solo un campione simbolico, rappresentativo del dominio dei possibili input.

1.2 Scopo del lavoro di tesi

Gli strumenti automatizzati per l'analisi statica della sicurezza hanno avuto una rapida evoluzione nell'ultimo decennio, partendo dalla semplice analisi lessicale fino ad integrare tutte le tecniche sopra descritte. Un primo obiettivo di questo lavoro di tesi è dunque quello di individuare un set di potenziali strumenti per l'analisi statica della sicurezza, che soddisfino determinati requisiti che saranno approfonditi nel prossimo Capitolo.
Bisogna infatti tenere in considerazione che l'insieme degli strumenti automatizzati è molto vario al suo interno: questi possono essere designati a uno specifico linguaggio o a più linguaggi di programmazione; possono essere integrati in un qualche IDE come plugin, o fare parte di una qualche piattaforma di CI/CD, o essere strumenti stand-alone da linea di comando. Gli strumenti possono ancora differenziarsi per tipo di licenza (commerciale o gratuita), per tipologia di vulnerabilità individuabili,

per modalità di presentazione dei report post-analisi, per velocità di esecuzione e scalabilità.

Lo scopo finale è quello di fornire una valutazione dei suddetti strumenti, ragione per cui si rende necessario individuare delle applicazioni che possano fungere da code base. Ci sono stati negli anni alcuni tentativi di ideare dei Benchmark che potessero essere utili nella creazione di una letteratura in materia di analisi statica della sicurezza: i due principali sono stati l'OWASP Benchmark project [OWASP Foundation (2020)] [4] e la Juliet Test Suite [National Institute of Standards and Technology (2005-2017)] [5]. In questo lavoro di tesi si ritiene però opportuno provare a valutare la sicurezza nell'ambito di applicazioni distribuite, di vario genere e scopo, che non siano state progettate come batterie di test per la SAST. Il secondo obiettivo di questa tesi è dunque quello di individuare, tramite il portale Github, delle applicazioni Open Source che soddisfino determinati requisiti, i quali verranno anch'essi discussi nel prossimo Capitolo.

Una volta individuati degli strumenti e delle applicazioni adeguate, il successivo obiettivo è dunque quello di eseguire effettivamente la SAST Analysis e individuare un modello di raccolta dei dati forniti dalle interfacce grafiche, al fine di avere un quadro il più possibile uniforme e completo delle vulnerabilità individuate, della loro pericolosità e della loro categorizzazione. Inoltre, come discusso in precedenza, l'analisi statica necessita di una revisione manuale per comprendere se la vulnerabilità sia da intendere come un vero/falso positivo/negativo.

Infine, questo lavoro di tesi prevede che i dati raccolti vengano raggruppati e rielaborati, al fine di creare degli indici utili a valutare gli strumenti. Ciò prevede inoltre la scelta di un sistema di scoring, che possa anche fornire delle indicazioni per confrontare tra loro gli strumenti per quanto riguarda la loro accuratezza.

Capitolo 2

Applicazioni e strumenti utilizzati

In questo Capitolo verranno discusse le scelte compiute nella prima parte del lavoro di tesi, aventi come scopo quello di creare un setting preparatorio il più possibile aderente ad alcuni requisiti, in vista delle successive analisi. In particolare verranno analizzate le decisioni prese per quanto riguarda gli strumenti automatizzati adottati, le applicazioni prese in considerazione come code base e le configurazioni generali.

2.1 Scelta degli strumenti

Come accennato nel Capitolo 1, negli ultimi anni il panorama degli strumenti automatizzati per l'analisi statica è cresciuto esponenzialmente, e si è molto diversificato al suo interno. Alcuni importanti enti promotori della sicurezza del software e della cybersecurity in generale, pubblicano periodicamente sui loro siti web delle liste di strumenti automatizzati, utili per l'analisi statica o dinamica; la ricerca è dunque partita da questo tipo di pubblicazioni, tra le quali vale la pena citare:

- **"Source Code Analysis Tools"** di OWASP Foundation [6]: l'Open Web Application Security Project è un'organizzazione no-profit che mira a diffondere buone pratiche di programmazione sicura.

- **"Source Code Security Analyzers"** di NIST [7]: il National Institute of Standards and Technologies è un'agenzia governativa americana che si occupa di regolamentare e gestire le nuove tecnologie.

2.1.1 Criteri principali di selezione

Partendo da questa base e conducendo ulteriori ricerche online, si è quindi individuato un cospicuo range di potenziali strumenti, molto differenti fra loro: al fine di poter raggiungere un numero compreso tra cinque e dieci strumenti, si è ritenuto opportuno effettuare una selezione in base a quattro criteri principali, e contemporaneamente a prendere in considerazione altre caratteristiche secondarie. I quattro criteri principali presi in considerazione sono stati:

Linguaggio

Gli strumenti di analisi non sono indipendenti dal tipo di programmazione utilizzato dal codice, poiché le regole da verificare sono pensate specificatamente per uno o più linguaggi: da ciò deriva un'abbondante presenza di strumenti per i linguaggi più diffusi o popolari (come Java, Javascript, C, C++...) e, al contrario, una difficoltà di reperibilità per quelli meno noti. In questo lavoro di tesi si è scelto di procedere adottando strumenti che presentassero un numero sufficientemente ampio di regole per il linguaggio Java, al fine di garantire uno spettro più ampio di possibili scelte.

Integrazione

Alcuni degli strumenti sono facilmente integrabili in uno o più IDE (Integrated Development Environment) tramite plugin scaricabile dagli appositi Marketplaces; altri sono inseriti in qualche piattaforma di gestione di pipelines CI/CD (Continuous Integration/Continuous Delivery); altri strumenti sono infine fruibili come standalone, ossia scaricabili ed eseguibili da linea di comando. A volte lo stesso strumento è disponibile in più versioni di integrazione, ma la diversa modalità può influire molto sulle tipologie di vulnerabilità individuabili, ragione per cui si è ritenuto di scegliere un'unica modalità di integrazione per tutti gli strumenti, per poter così favorire un confronto più equo. In particolare, si è scelto di procedere con strumenti che fossero disponibili come plugin per almeno uno dei seguenti IDE: Eclipse, Visual Studio Code o IntelliJ IDEA.

Licenza

La maggior parte degli strumenti presenti sul mercato prevede il pagamento di una licenza commerciale, in genere destinata ad organizzazioni aziendali più o meno grandi, che desiderano tutelare la sicurezza del loro codice. Ci sono tuttavia alcune delle aziende produttrici di strumenti che prevedono delle licenze gratuite per una versione di prova (generalmente della durata di poche settimane), o che consentono di usufruire di licenze educational (designate a scopi didattici e di ricerca). Altre

aziende consentono poi l'utilizzo gratuito dello strumento purché non si superi un certo numero di scansioni giornaliere o di linee di codice analizzate, o ne prevedono un uso libero ma al solo fine di valutare codice Open Source pubblico. Ci sono infine degli strumenti ad uso totalmente gratuito e libero, rilasciati sotto GNU Lesser General Public License. In questo lavoro di tesi sono stati chiaramente privilegiati questi ultimi strumenti gratuiti, ma sono stati inclusi anche altri strumenti ottenibili con le varie licenze commerciali speciali di cui sopra.

Vulnerabilità individuabili

Tutti gli strumenti sono corredati da una documentazione che ne attesta le effettive capacità, indicando quali sono le vulnerabilità individuabili e come queste vengano classificate in una qualche scala di pericolosità. In quasi tutti i casi alla vulnerabilità sono associati uno o più codici CWE: la Common Weakness Enumeration [8] è un dizionario ufficiale delle vulnerabilità software e hardware, mantenuto e aggiornato costantemente dall'organizzazione americana no-profit per la cybersecurity MITRE Corporation. Alcune di queste vulnerabilità presenti nella CWE vengono incluse ogni anno nelle classifiche delle vulnerabilità più pericolose; tra le organizzazioni che rilasciano questi ranking si possono nominare il SANS Institute (SysAdmin, Audit, Networking and Security, che si occupa di formare professionisti di cybersicurezza) o la sopracitata OWASP Foundation. In questo lavoro di tesi si è cercato di prediligere strumenti che fossero in grado di individuare, in maniera totale o parziale, le vulnerabilità presenti nel SANS25 (ossia le 25 più pericolose secondo il SANS [9]) e/o quelle dell' OWASP Top-10 (versione 2021 [10]).

2.1.2 Ulteriori caratteristiche preferenziali

Oltre ai quattro criteri di selezione di cui sopra, sono state inoltre valutate altre caratteristiche che, sebbene non indispensabili, sono state considerate preferenziali nella scelta degli strumenti, a parità delle precedenti:

- Personalizzazione: la possibilità di selezionare solo alcune delle regole elencate dallo strumento è da considerarsi come un vantaggio, poiché permette di eseguire delle analisi on-the-fly filtrando per tipologia di vulnerabilità (es: solo vulnerabilità di sicurezza vere e proprie, e non bugs), per pericolosità (es: solo vulnerabilità critiche), per grado di confidenza (es: solo confidenza alta), riducendo sensibilmente il tempo di esecuzione. La possibilità di disattivare permanentemente alcune regole dall'analisi consente inoltre di evitare massicce quantità di segnalazioni nei report, a fronte di vulnerabilità considerate solo marginalmente pericolose.
- Interfaccia dei risultati: gli strumenti generalmente prevedono delle interfacce grafiche per presentare i report dei risultati, che possono essere più o meno

complete e più o meno interattive. Sono stati privilegiati gli strumenti che marcassero chiaramente, ad esempio con colorazioni differenti, i diversi gradi di pericolosità, o che permettessero di raggruppare o filtrare i risultati per vulnerabilità o per pericolosità. Sono stati considerate privilegiate anche quelle interfacce che prevedono una chiara descrizione della vulnerabilità, con i corrispondenti riferimenti CWE e con degli eventuali suggerimenti per la correzione.

- Velocità di esecuzione e scalabilità: seppure non misurate analiticamente, anche queste due caratteristiche sono state tenute in considerazione nella scelta degli strumenti. Bisogna infatti tenere presente che il tempo di esecuzione aumenta in modo non lineare a seconda che il contesto di analisi sia a livello di linea di codice, di funzione, di modulo, o di intera applicazione. Inoltre, considerando che le LOC (lines of code) delle applicazioni selezionate possono differire di un intero ordine di grandezza, sono stati considerati vantaggiosi quegli strumenti che, al crescere dell'applicazione, non mostravano particolari rallentamenti nell'elaborazione dell'analisi.

2.1.3 Strumenti adottati

Nella Tabella 2.1 sono stati inseriti tutti gli strumenti di analisi statica considerati nella fase di valutazione, ordinati alfabeticamente e tenendo presente che essi soddisfano già tutti in partenza il requisito della compatibilità col linguaggio Java. Nella seconda colonna di questa tabella riepilogativa gli strumenti sono stati classificati in base alla disponibilità o meno di un plugin, non tenendo conto di altre forme di integrazione; nella terza colonna si è voluto sottolineare la natura gratuita dello strumento (ossia ne esiste almeno una versione basilare che sia gratuita) o commerciale (in cui sono tuttavia incluse anche le licenze gratuite speciali di cui discusso nel Paragrafo 2.1.1). Infine nella quarta colonna si sono indicati gli standard a cui gli strumenti si attengono, stando alla documentazione; si tenga tuttavia presente che tali standard potrebbero non essere soddisfatti in tutte le versioni dello strumento, ma valere ad esempio solo per alcune tipologie di integrazione, o solo con la versione di licenza commerciale standard riservata alle aziende; quelli indicati sono dunque gli standard massimi che si possono raggiungere con una qualche versione dello strumento.

Dei 23 strumenti elencati nella Tabella 2.1, in questo lavoro di tesi ne sono stati selezionati sette, in base ai criteri discussi nel Paragrafo 2.1.1. Considerando che tra di essi sono presenti anche alcuni prodotti commerciali, si è tuttavia ritenuto opportuno procedere non evidenziando in maniera esplicita i nomi dei vari strumenti, ma pseudonomizzandoli con "Strumento 1", "Strumento 2" e così via,

fino ad arrivare a "Strumento 7", etichette con cui ci si riferirà agli strumenti da questo momento in avanti.

Strumento	**Integrazione**	**Licenza**	**Standards**
CheckStyle	Plugin	Gratuito	
Codacy		Commerciale	OWASP Top 10
CodeSonar	Plugin	Commerciale	OWASP Top 10 SANS 25
Code Sight	Plugin	Gratuito	
CxSAST	Plugin	Commerciale	OWASP Top 10 SANS 25
DeepSource		Commerciale	OWASP Top 10
Embold	Plugin	Gratuito	OWASP Top 10
Fortify	Plugin	Commerciale	Valutazione con OWASP benchmark
Horusec	Plugin	Gratuito	
Kiuwan	Plugin	Commerciale	OWASP Top 10 SANS 25
Klocwork	Plugin	Commerciale	OWASP Top 10 SANS 25
ParaSoft Jtest	Plugin	Commerciale	OWASP Top 10 SANS 25
PMD	Plugin	Gratuito	
PVS-Studio	Plugin	Commerciale	
Reshift		Commerciale	
Semgrep		Commerciale	OWASP Top 10
ShiftLeft NG		Commerciale	OWASP Top 10
Sigma	Plugin	Commerciale	
SonarLint	Plugin	Gratuito	OWASP Top 10 SANS 25
Snyk	Plugin	Gratuito	OWASP Top 10 SANS 25
Sparrow	Plugin	Commerciale	OWASP Top 10 SANS 25
Spotbugs	Plugin	Gratuito	OWASP Top 10
Xanitizer		Commerciale	OWASP Top 10 SANS 25

Tabella 2.1: Strumenti per l'analisi statica compatibili con il linguaggio Java

2.2 Applicazioni utilizzate

In questa sezione si discuterà del secondo elemento fondamentale per la creazione di un setting preparatorio adeguato: la code base. In questo caso la base dati per la ricerca di potenziali applicazioni è stato il servizio di hosting online per progetti software Github: grazie alle sue funzionalità di filtraggio, ad esempio in base a delle key-words predefinite, è stato infatti possibile condurre la selezione in modo più agevole rispetto ad altre piattaforme.

2.2.1 Requisiti e criteri di scelta

Anche in questo caso è stato necessario stabilire tre criteri di selezione esclusiva (che hanno portato a un risultato di 55 potenziali applicazioni) e cinque criteri secondari, che hanno permesso di effettuare un ulteriore filtraggio.

Criteri principali

- Linguaggio lato back-end: come già accennato nella precedente sezione, si è deciso di analizzare solo applicazioni che presentassero il lato back-end scritto totalmente in Java: ciò permette da un lato di non restringere troppo il campo di ricerca, e dall'altro di avere a disposizione un range più vario di strumenti di analisi compatibili.

- Framework applicativo: al fine di ottenere una maggiore omogeneità nelle categorie di vulnerabilità individuabili, e favorire così il confronto tra gli strumenti, si sono selezionate esclusivamente applicazioni che adottassero il framework Spring Boot, specifico modulo dello Spring Framework che semplifica e accelera lo sviluppo di applicazioni web, riducendone la complessità di configurazione.

- Architettura: l'obiettivo di questa tesi era quello di testare il funzionamento degli strumenti per l' analisi statica su applicazioni distribuite, motivo per cui nella selezione sono state escluse le applicazioni locali e sono state invece incluse le classiche architetture a due livelli di tipo client-server, tenendo comunque in considerazione eventualmente anche architetture a tre o più livelli di distribuzione.

Criteri secondari

Le applicazioni così filtrate sono state poi oggetto di un'ulteriore classificazione utile ad assegnare un punteggio di priorità, in modo da poter analizzare le applicazioni più significative e allo stesso tempo selezionarne un campione sufficientemente vario

e completo. Il punteggio è stato assegnato in modo che a uno score più basso corrispondesse una priorità più alta. Le caratteristiche oggetto di classificazione sono state:

- Analisi preliminari: in questa fase sono state condotte delle analisi preliminari delle vulnerabilità, di tipo più quantitativo che qualitativo, al fine di assegnare una priorità minore a quelle applicazioni che presentassero già ai primi test pochissime vulnerabilità, o che ne presentassero alcune, ma di pericolosità molto bassa. Questo è stato il criterio preponderante ai fini dell'assegnazione del punteggio.

- Linguaggio lato front-end: si è fatto in modo di includere sia alcune applicazioni che presentassero il lato front-end scritto in Java, sia altre il cui front-end fosse invece scritto in linguaggi come Javascript o HTML.

- Architettura a microservizi: si è scelto di selezionare alcune applicazioni che fossero strutturate in microservizi, ma anche di includerne altre che fossero invece monolitiche.

- Contesto applicativo: le applicazioni selezionate provengono da contesti molto diversi fra loro, dagli store online, al campo medico, alla gestione aziendale. Quelle selezionate sono dunque applicazioni comuni, non appositamente scritte in vista di potenziali test di analisi della sicurezza. Tuttavia, è stata inclusa anche un'applicazione volutamente vulnerabile: si tratta di VulnerableApp, un progetto della OWASP Foundation che comprende al suo interno diverse categorie di vulnerabilità e si presenta con una struttura modulare e scalabile.

- LOC (Lines Of Code): le applicazioni selezionate variano molto in quanto a numero di linee di codice, poiché si è cercato di includere sia progetti più estesi, intorno alle 350K LOC, sia piccole applicazioni di circa 10K LOC.

2.2.2 Applicazioni adottate

Punteggio	**Priorità**	**Numero di Applicazioni**
1	Alta	7
2	Media	9
3	Bassa	10
4	Bassissima	29

Tabella 2.2: Scala di punteggi delle priorità

Al termine di questa fase di classificazione, è stato quindi assegnato alle applicazioni un punteggio di priorità da 1 a 4, come si evince dalla Tabella 2.2.

Si è in seguito deciso di escludere le applicazioni a priorità bassissima, mentre quelle a priorità bassa vengono elencate qui per completezza pur non essendo state parte dell'analisi definitiva, che ha invece incluso le sette applicazioni a priorità alta e, come riserva aggiuntiva, le nove applicazioni a priorità media. Nella Tabella 2.3 si elencano le 26 applicazioni a priorità alta, media o bassa, ordinate per numero di linee di codice descrescente.

Applicazione	**Analisi Preliminari**	**Front-end**	**Micro servizi**	**Ambito**	**LOC**	**Punteggio**
cwa-server [11]	Ottime	Java	Sì	Medico	355.9K	1
shopizer	Sufficienti	Java	No	Shopping	334K	3
mycollab [12]	Buone	Java	No	Gestionale	300K	2
moviebuffs	Sufficienti	Java	No	Shopping	124K	3
web-application-for-donors [13]	Buone	Altri	No	Medico	122K	2
21-points	Sufficienti	Altri	No	Medico	101K	3
BookStoreApp-Distributed-Application [14]	Buone	Altri	Sì	Shopping	69.3K	2
suricate	Sufficienti	Altri	No	Gestionale	47.7K	3
yugastore-java [15]	Ottime	Java	Sì	Shopping	45.7K	1
spring-security-react-ant-design-polls-app	Sufficienti	Altri	No	Sondaggi	44K	3
atsea-sample-shop-app [16]	Buone	Altri	No	Shopping	41.7K	2
employee-management [17]	Ottime	Java	No	Gestionale	38.1K	1
my-moments [18]	Buone	Java	Sì	Social Media	32.1K	2
Appointment Scheduler [19]	Buone	Java	No	Gestionale	31.6K	2
Poplar [20]	Ottime	Altri	Sì	Social Media	27.8K	1
TravelWeb ApplicationVirtugo [21]	Buone	Altri	No	Tempo Libero	26.3K	2
MyHome	Sufficienti	Java	No	Shopping	26.2K	3
book-project [22]	Buone	Java	No	Shopping	26.1K	2
jcart [23]	Ottime	Altri	No	Shopping	22.6K	1
document-management-store-app [24]	Ottime	Java	No	Gestionale	21.1K	1
spring-boot-react-blog [25]	Buone	Java	No	Social Media	17.2K	2
spring-petclinic	Sufficienti	Java	No	Gestionale	14K	3
VulnerableApp [26]	Ottime	Java	No	Vulnerabilità	11.3K	1
piggymetrics	Sufficienti	Altri	Sì	Gestionale	8.3K	3
Ward	Sufficienti	Altri	No	Gestionale	6K	3
microservices-springboot	Sufficienti	Java	Sì	Shopping	3.3K	3

Tabella 2.3: Applicazioni con punteggio di priorità da 1 a 3

2.3 Configurazioni

In questa breve sezione si elencheranno le configurazioni generali utilizzate per creare l'ambiente preparatorio alle analisi, che saranno discusse nel successivo capitolo.

Sistema Operativo

Il sistema operativo adottato è stato Linux 5.4.0-89-generic, configurato su una macchina virtuale con distribuzione Ubuntu 20.04.03 LTS VM.

Java

Per quanto riguarda l'implementazione Java, è stata scelta la versione OpenJDK 11.0.11.

IDEs

Sono stati scelti tre IDEs:

- Eclipse IDE for Enterprise Java and Web Developers 2021-09 (4.21.0): uno dei più popolari e diffusi IDEs sul mercato, vanta un ricco Marketplace di plugins e un'ottima integrazione con Apache Maven.
- IntelliJ IDEA Community Edition 2021.3: anch'esso molto diffuso e fortemente personalizzabile tramite plugins, fornisce una migliore intergrazione con il sistema di build Gradle.
- VSCode 1.62.2: meno diffuso in ambito Java e più limitato in quanto a funzionalità aggiuntive rispetto ai due precedenti, è tuttavia molto più leggero e veloce, e presenta un'ottima interfaccia grafica.

Altro

Sono stati anche utilizzati:

- Buildship Gradle Integration 3.0 plugin for Eclipse.
- SpringTools 4.12.1 plugin for Eclipse.
- Docker 20.10.10.
- Apache Maven 3.6.3.

Capitolo 3

Processo di Analisi delle Vulnerabilità

In questo terzo capitolo verrà discussa la metodologia utilizzata per condurre il vero e proprio processo di analisi statica della sicurezza: l'obiettivo preposto è quello di individuare potenziali vulnerabilità nel codice per poi effettuarne una revisione manuale, la quale si rende necessaria a causa della grande predisposizione alla segnalazioni di falsi positivi da parte degli strumenti automatizzati. Vi è inoltre la necessità di ideare un modello per raccogliere i dati così ottenuti,per poterne disporre in modo più agevole nella successiva fase di elaborazione che verrà trattata nei Capitoli 4 e 5. In questo Capitolo si procederà partendo dalla descrizione del modello adottato per la raccolta dati e del workflow generale del processo di analisi; si passerà quindi al processo di attribuzione manuale dei flag di veri e falsi positivi e negativi, per arrivare infine a delle considerazioni di carattere globale emerse durante le analisi.

3.1 Schema di raccolta dei dati

Nella prima fase del lavoro di analisi sono state considerate solo le sette applicazioni a priorità alta appartenenti alla Tabella 2.3 e i sette strumenti selezionati descritti nel Paragrafo 2.1.3; in generale, il processo di individuazione delle vulnerabilità è stato impostato in modo da analizzare un'applicazione per volta con ogni strumento. L'organizzazione più logica dei dati è stata quindi quella di utilizzare diversi fogli di calcolo, uno per ogni applicazione, nelle cui colonne vengono elencate una serie di informazioni:

- nome della vulnerabilità ed eventuali codici CWE, che servono a identificare in modo univoco la vulnerabilità.

- posizione della vulnerabilità all'interno del codice, utile per poter svolgere i passaggi successivi di revisione manuale senza la necessità di rieseguire l'analisi. Per posizione della vulnerabilità si intende il microservizio (indicato tra parentesi, qualora presente), il nome del file e il numero di linea di codice.

- eventuali note esplicative sulla natura della vulnerabilità

- due colonne per ognuno dei sette strumenti, di cui la prima per inserire un flag di vera/falsa positività/negatività alla vulnerabilità, la seconda per indicare con che grado di pericolosità essa è stata individuata dallo strumento; alle vulnerabilità di pericolosità medio-alta è stato attribuito il valore HIGH, a quelle di pericolosità medio-bassa, il valore LOW. Ulteriori indicazioni sull'attribuzione dei flag e dei valori di pericolosità verranno discusse nelle prossime Sezioni di questo Capitolo.

In Figura 3.1 viene illustrata a titolo di esempio una cattura del foglio di calcolo utilizzato per raccogliere i dati relativi a una delle applicazioni, "cwa-server". Si possono notare le colonne sopra citate, con le due colonne colorate per ciascuno strumento.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1	POTENTIAL VULNERABILITY	STANDARD	REFERENCE	NOTES														
2	Potential CRLF log injection	CWE-93 CWE-117	(callback)Callback(		TP	HIGH												
3	Potential path traversal	CWE-22	(distribution)Output	The path setter is never called	FP	HIGH			TN	HIGH	TN	HIGH					TN	HIGH
4	Unsafe hash comparison	CWE-203	(distribution)TestDi		TP	HIGH			FN	HIGH								
5	Unsafe hash comparison	CWE-203	(distribution)TestDi		TP	HIGH			FN	HIGH								
6	CSRF protection disabled	CWE-352	(submission)Secur		TP	HIGH					TP	HIGH	FN	LOW			TP	HIGH
7	Possible null pointer dereference	CWE-476	(callback)Registrat		TP	LOW	TP	HIGH			FN	LOW						
8	Not injected member of Spring		(distribution)Digit				TP	HIGH										
9	Not injected member of Spring		(distribution)Faile				TP	HIGH										
10	Not injected member of Spring		(distribution)S3Rete				TP	HIGH										
11	Not injected member of Spring		(distribution)S3Rete				TP	HIGH										
12	Not injected member of Spring		(distribution)S3Rete				TP	HIGH										
13	Not injected member of Spring		(distribution)TestD				TP	HIGH										
14	Incorrect equality testing for floating point		(submission)EventN	Never called method	TN	HIGH	FP	HIGH										
15	Not injected member of Spring		(submission)FakeC				TP	HIGH										
16	Not injected member of Spring		(upload)TestDataG				TP	HIGH										
17	Weak hash function hashCode	CWE-328	(distribution)Validat						TP	HIGH								
18	Weak hash function hashCode	CWE-328	(distribution)Valida						TP	HIGH								
19	Weak hash function hashCode	CWE-328	(distribution)S3Obje						TP	HIGH								
20	Weak hash function hashCode	CWE-328	(distribution)TestD						TP	HIGH								
21	Unsafe hash comparison	CWE-203	(distribution)S3Obje		FN	HIGH			TP	HIGH								
22	Unsafe hash comparison	CWE-203	(distribution)Publis		FN	HIGH			TP	HIGH								
23	Potential SQL injection	CWE-89	(distribution)Objec	There is not any SQL code	TN	HIGH			FP	HIGH								
24	Weak hash function hashCode	CWE-328	(submission)Tan.ja						TP	HIGH								
25	Weak hash function hashCode	CWE-328	(download)BatchDo						TP	HIGH								
26	Weak TLS verification	CWE-295	(federation)Default		FN	HIGH			TP	HIGH								
27	Weak TLS verification	CWE-295	(federation)NoopHo	This component is never instanciat	TN	HIGH			FP	HIGH								
28	Weak hash function hashCode	CWE-328	(persistence)Diagn						TP	HIGH								
29	Weak hash function hashCode	CWE-328	(persistence)Federa						TP	HIGH								
30	Weak hash function hashCode	CWE-328	(persistence)Feder						TP	HIGH								
31	Potential SQL injection	CWE-89	(persistence)Feder	There is not any SQL code	TN	HIGH			FP	HIGH								
32	Resource leak (not closed)	CWE-459 CWE-404	(distribution)DscLi	It is a ByteArrayInputStream, no r	TN	LOW	TN	HIGH									FP	HIGH

Figura 3.1: Cattura del Foglio di calcolo utilizzato per la raccolta dei dati per l'applicazione "cwa-server"

3.2 Procedura generale di Analisi

In questa Sezione verrà descritto il workflow generale adottato durante la fase del lavoro di analisi; esso verrà suddiviso in più operazioni, che saranno inoltre successivamente approfondite nei prossimi Paragrafi.

Configurazione degli strumenti

Nell'approcciare il lavoro di analisi statica, la prima operazione da effettuare è stata, quando possibile, una configurazione delle regole dello strumento in modo da poter personalizzare gli output; possibili selezioni considerate sono state ad esempio in base alla pericolosità (in modo da escludere vulnerabilità estremamente poco significative dal punto di vista della sicurezza del codice) o in base alla tipologia di vulnerabilità.

Esecuzione dell'analisi

A questo punto è stata lanciata l'esecuzione dell'analisi statica automatizzata, su tutto il progetto qualora si trattasse di un'applicazione monolitica, o su un microservizio per volta, nei casi in cui invece l'applicazione prevedesse dei microservizi. Dai risultati sono state escluse le vulnerabilità individuate al di fuori della cartella principale contenente il sorgente Java, ossia la classica "./src/main/java" o equivalenti: in questo modo, anche qualora lo strumento avesse potuto individuare vulnerabilità anche in altri linguaggi, queste non sono state elencate nei risultati; inoltre, ciò ha permesso di non tenere conto delle vulnerabilità, quasi sempre evidentemente volute, nella cartella dei sorgenti di test. Sono state inoltre scartate, qualora non fosse stato possibile con una configurazione iniziale dello strumento, tutte quelle segnalazioni che non consistessero in vere e proprie vulnerabilità di sicurezza, ma in problematiche minori come bugs di lievissima entità, warnings riguardanti lo stile del codice, consigli per la rinominazione delle variabili e così via.

Registrazione dei dati

I risultati così ottenuti sono stati poi inseriti nei fogli di calcolo di cui alla Sezione 3.1; in prima battuta sono state inserite solo le vulnerabilità individuate e classificate come di pericolosità medio-alta da almeno uno strumento, indicando il valore HIGH nella seconda colonna di ciascuno strumento che l'avesse individuata ed elencata come altamente pericolosa. In seguito sono stati aggiunti per quelle stesse vulnerabilità anche i valori LOW nelle colonne di quegli strumenti che, pur avendo individuato la vulnerabilità, l'avessero classificata come poco pericolosa. Non sono state aggiunte ulteriori vulnerabilità, al fine di escludere quelle segnalazioni mai classificate come molto pericolose da nessuno strumento.

Revisione manuale

A questo punto si è proceduto con la revisione manuale delle vulnerabilità, al fine di valutare se quelli individuati fossero dei casi di veri o falsi positivi, operando in base a determinate motivazioni come quelle riportate nel Paragrafo 3.3.1. Infine si

è concluso il lavoro di analisi considerando i casi negativi, ossia le vulnerabilità non individuate dagli strumenti, secondo la metodologia descritta nel Paragrafo 3.3.2.

Ulteriori analisi

Una volta svolta l'analisi principale con le sette applicazioni a priorità alta e i sette strumenti selezionati, si è proceduto anche a un'analisi secondaria utilizzando le nove apps a priorità media elencate nella Tabella 2.3. Queste applicazioni sono state trattate in maniera leggermente differente, poiché sono state considerate come un'unica applicazione chiamata "others", e dunque le vulnerabilità raccolte per ciascuna di essa sono state raggruppate in un unico foglio di calcolo. Un'altra differenza sostanziale con le analisi di cui sopra consiste nel fatto che le uniche vulnerabilità inserite sono state quelle rientranti nell' OWASP Top 10, versione 2021 [10]: avendo infatti raggiunto un numero sostanzialmente corposo di vulnerabilità di pericolosità medio-bassa, sufficiente a condurre delle buone elaborazioni, si è deciso di utilizzare le applicazioni in riserva solo per approfondire la presenza di vulnerabilità di pericolosità alta, al fine di ottenere dei risultati finali più significativi, come verrà discusso nel Paragrafo 4.1.3

3.3 Attribuzione dei flag valutativi

3.3.1 Valutazione di vera o falsa positività

Considerato l'alto numero di segnalazioni che costituiscono falsi positivi, tipico di qualsiasi procedura di analisi statica della sicurezza, in seguito alle procedure automatizzate è stata effettuata una revisione manuale dei risultati. A tutte le vulnerabilità individuate secondo la metodologie descritta nella precedente sezione, è stato inizialmente assegnato un flag TP provvisorio, stante a significare che la vulnerabilità individuata dallo strumento è coincidente con una reale vulnerabilità. In seguito, ogni segnalazione è stata analizzata singolarmente, e alcune di esse sono state in realtà etichettate come FP, ossia come dei falsi positivi. Vengono di seguito elencate le motivazioni più comuni che hanno spinto all'inversione di flag durante la revisione manuale:

- **Vulnerabilità inesistente**: caso piuttosto raro, in cui lo strumento riporta una segnalazione del tutto incoerente con il codice sorgente. Ad esempio, viene segnalata una SQL Injection a fronte di un codice non contenente affatto istruzioni SQL, o facente riferimento a delle linee di codice vuote o commentate.
- **Componente mai istanziato**: la vulnerabilità è effettivamente presente, ma dato che lo specifico componente Spring non viene mai istanziato all'interno del resto del codice, ciò non costituisce un reale pericolo per l'applicazione.

- **Metodo mai chiamato**: è effettivamente presente una vulnerabilità all'interno dello specifico metodo, tuttavia esso fa parte del cosiddetto dead code, ossia codice non raggiungibile perché mai chiamato da nessun altro metodo; anche in questo caso non si tratta quindi di una reale minaccia. Questo genere di vulnerabilità non sarebbero individuate effettuando un'analisi dinamica della sicurezza del codice.

- **Chiusura della risorsa non necessaria**: in alcuni casi, la mancata chiusura di una risorsa (ad esempio, un ByteArrayInputStream) non costituisce una vulnerabilità, poiché trattasi di oggetti che vengono chiusi automaticamente senza bisogno di una chiamata esplicita alla close() da parte dello sviluppatore.

- **Risorsa chiusa dal chiamante**: la segnalazione di una mancata chiusura di un oggetto risulta a volte non veritiera, poiché l'oggetto in questione risale nello stack di chiamate, venendo passato come parametro al metodo padre, che si occupa di chiuderlo.

- **Controllo sugli Optional presente**: può capitare che lo strumento riporti un mancato controllo sull'effettiva presenza di un oggetto di tipo Optional, da effettuarsi con il corrispettivo metodo isPresent(). In alcuni casi tuttavia, anche se il controllo non viene effettivamente effettuato nella stessa linea di codice in cui l'oggetto è utilizzato, esso è presente all'interno di una precedente istruzione condizionale e dunque, in mancanza dell'oggetto opzionale, il frammento di codice non verrebbe eseguito e non vi sarebbe nessun reale pericolo.

- **Falsa SQL injection**: saltuariamente gli strumenti riportano una vulnerabilità di tipo SQL Injection, mentre in realtà i parametri passati alla query SQL sono delle costanti, non modificabili dagli utenti e dunque non soggetti a potenziali iniezioni di codice malevolo.

Va ad ogni modo sottolineato come quelli sopracitati siano solo esempi relativi alle casistiche *più frequenti* di rilevazione di falsa positività: sono state tuttavia riscontrate, seppur più raramente, ulteriori tipologie di vulnerabilità oggetto di false segnalazioni, oltre a vari fattori che potessero causare le stesse vulnerabilità qui esposte. Citando ad esempio le SQL Injections, si può affermare che la non modificabilità dei parametri sia la causa più frequente di falsi positivi, ma non di certo l'unica, essendo stati rilevati anche casi di prepared statements non riconosciuti.

Va allo stesso modo sottolineato come anche l'elenco completo possa non essere esaustivo poiché, in un caso più generale con un campione maggiore, varie altre vulnerabilità potrebbero dare luogo a falsi positivi: riprendendo ancora il precedente esempio, alle SQL Injections potrebbero aggiungersi altre categorie di Injections come le XSS o la LDAP.

3.3.2 Individuazione delle negatività

La successiva fase della revisione manuale dei risultati, dopo l'attribuzione del flag di vera o falsa positività, è stata quella del passaggio all'attribuzione dei flags di vera o falsa negatività: si è trattato dunque di gestire quei casi in cui nello specifico frammento di codice non è stata segnalata alcuna vulnerabilità, talvolta a ragione poiché effettivamente si trattava di codice sicuro, e talvolta in errore, poiché in realtà una vulnerabilità era presente. Naturalmente, gli strumenti riportano nella segnalazioni soltanto quelle situazione che essi interpretano come vulnerabilità, e dunque tra i risultati forniti non sono presenti dei casi di negatività; d'altro canto, il totale della code base considerata ammonta a più di un milione di linee di codice, motivo per cui una revisione manuale di ogni singola LOC alla ricerca di potenziali vulnerabilità è stata considerata una soluzione non percorribile.

Si è optato dunque per una metodologia intermedia per individuare i casi di negatività, confrontando le segnalazioni dei vari strumenti. Qualora almeno uno strumento avesse individuato una vulnerabilità a cui, in seguito alla fase di revisione, è stato attribuito un flag di true positive, è chiaro che per gli strumenti non in grado di individuarla ciò si possa tradurre in un flag di false negative (FN). Viceversa, qualora almeno uno strumento avesse individuato una vulnerabilità a cui viene attribuito un flag di false positive, per gli strumenti che (in maniera corretta) non avessero riportato alcuna segnalazione, ciò si può tradurre in un flag di true negative (TN).
In realtà prima di attribuire i flags di negatività, fortemente determinanti nella successiva valutazione degli strumenti, è stata condotta una verifica preliminare sulle rispettive documentazioni ufficiali: nel caso in cui lo strumento (per sua natura o in base alla versione di integrazione e alla licenza scelte) non risultasse in grado di individuare affatto quella specifica tipologia di vulnerabilità tra le sue regole, è chiaro che assegnare un flag di negatività impatterebbe in modo non meritocratico sulla sua valutazione. E' infatti corretto valutare uno strumento in base al fatto che avrebbe potuto, stando alle sue regole, individuare una vulnerabilità ma non lo ha fatto, sia nel caso di vulnerabilità presente (caso false negative), sia quando questa effettivamente non era presente (caso true negative).

Ne consegue che le vulnerabilità per cui si sono registrati più frequentemente casi di negatività sono state quelle potenzialmente individuabili da più di uno strumento; se ne riportano di seguito le principali categorie, coinvolte sia in casi di vera che di falsa negatività:

- SQL injections (sia di tipo semplice che di tipo JDBC)

- Mancate chiusure di risorse (Resource Leak)

- Vulnerabilità legate al framework Spring (come Persistent Entity in @Request-Mapping o Not injected member of Spring)

- Vulnerabilità legate alle funzioni di Hash (Unsafe Hash comparison principalmente, ma anche Weak Hash Function).

3.4 Spettro delle vulnerabilità individuate

L'obiettivo di questa sezione è quello di fornire una vista di insieme delle vulnerabilità individuate durante il processo di analisi, approfondendo le principali minacce e studiandone le caratteristiche, al fine di poter elaborare delle statistiche sullo stato della sicurezza delle applicazioni che fungono da code base.

3.4.1 Statistiche di ordine generale

Per poter fornire una visione di insieme delle minacce presenti, viene presentata la Tabella 3.1, in cui vengono elencate in ordine alfabetico tutte le tipologie di vulnerabilità raccolte nella fase di analisi: si tratta, come precedentemente discusso nella Sezione 3.2, di tutte e sole quelle possibili segnalazioni categorizzate come di pericolosità alta da almeno uno strumento. Si può dunque partire analizzando il volume totale dei risultati raccolti, in termini sia di vulnerabilità che di occorrenze delle stesse:

- 251 vulnerabilità in totale nella code base
- 37 diverse tipologie di vulnerabilità

Sempre in Tabella 3.1 viene inoltre fornita un'identificazione della vulnerabilità, costituita da uno o più codici tratti dalla Common Weakness Enumeration; si è scelto di includere solo i codici forniti dalle interfacce dei vari strumenti; in genere queste ultime risultavano coincidenti tra loro, ma in alcuni casi è stato necessaria una unione di più codici diversi ma ugualmente validi. In alcuni rari casi, non è stato possibile assegnare un'etichetta CWE, poiché trattasi di vulnerabilità troppo specifiche, o viceversa troppo generiche.

VULNERABILITY NAME	CWE	OWASP	CWA-SERVER	DM_STORE	EMPLOYEE	JCART	POPLAR	VULNERABLE	YUGASTORE
Always false expression	CWE-570						x		
Always true expression	CWE-571						x	x	
Bad exadecimal concatenation	CWE-704				x				
Command Injection	CWE-78	A03						x	
CSRF Protection disabled	CWE-352	A01	x	x		x			x
Dynamically loaded class	CWE-470	A03		x					
Empty catch block	CWE-1069			x					
Hard coded password	CWE-259	A07						x	
Impossible cast							x	x	
Incorrect equality testing for floating point			x						
Information exposure through error	CWE-209	A04				x			x
IV should be randomized	CWE-329 CWE-330	A02			x				
Not injected member of Spring			x	x	x	x	x		
Null pointer dereference	CWE-476								x
Optional accessed without “isPresent”	CWE-476 CWE-457				x			x	x
Persistent Entity in @RequestMapping	CWE-915	A08			x	x			x
Possible Null pointer dereference	CWE-476		x					x	
Possible ThreadLeak			x						
Potential CRLF log injection	CWE-93 CWE-117	A03 A09	x				x		
Potential JDBC SQL injection	CWE-89	A03		x				x	
Potential LDAP injection	CWE-90	A03				x			
Potential path traversal	CWE-22	A01	x	x				x	
Potential SQL injection	CWE-89	A03	x		x		x		x
Potential SSRF	CWE-918 CWE-73	A10 A04						x	
Property leak	CWE-212 CWE-213	A04			x				
Reference used, then verified against null	CWE-476			x					
Regex DOS	CWE-400				x				
@RequestMapping without http method	CWE-352	A01		x	x	x	x	x	x
Resource leak (not closed)	CWE-459 CWE-404		x	x		x		x	
Return value not used					x				
Sensitive information in tmp file	CWE-276	A01		x					
Unsafe hash comparison	CWE-203		x						
Useless @Scope annotation									x
Weak hash function	CWE-328	A02			x		x		
Weak hash function hashCode	CWE-328	A02	x	x				x	x
Weak random method	CWE-330	A02						x	
Weak TLS verification	CWE-295	A07	x						

Tabella 3.1: Vulnerabilità individuate nelle applicazioni a priorità alta

3.4.2 Statistiche sulle sole vulnerabilità OWASP

Nella Tabella 3.1 viene anche riportato un eventuale codice OWASP: si tratta di un'etichetta variabile tra A01 e A10 e indica che la vulnerabilità, in base al codice o ai codici CWE che le sono stati assegnati, rientra nella top 10 delle vulnerabilità più pericolose secondo l'OWASP, aggiornata alla versione del 2021 [10]. In realtà, tra le vulnerabilità individuate nella code base in oggetto non sono emersi esempi riguardanti tutte e dieci le vulnerabilità in cima al ranking, ma solo parte di esse, delle quali viene qui fornita una breve descrizione.

A01:2021-Broken Access Control

Consiste nella violazione del controllo degli accessi di un servizio, che può portare alla divulgazione di dati sensibili al di fuori dei limiti previsti per gli utenti.

A02:2021-Cryptographic Failures

Consiste nella mancata applicazione di procedure utili a garantire la segretezza e l'integrità di dati più o meno sensibili, tramite operazioni crittografiche.

A03:2021-Injection

Le injections sono una famiglia di vulnerabilità che prevedono un'intrusione di codice malevolo da parte dell'utente in vari modi; si ha un'effettiva vulnerabilità quando gli input non vengono ripuliti e validati all'interno dell'applicazione.

A04:2021-Insecure Design

Consiste nella mancanza di un approccio corretto alla sicurezza informatica al livello di design dell'applicazione: non prevedendo determinati rischi informatici già in questa fase, non è spesso poi possibile correggere a posteriori alcune vulnerabilità.

A07:2021-Identification and Authentication Failures

Sono vulnerabilità connesse alla cattiva gestione del login utente e all'autenticazione, e dunque anche alla resistenza delle passwords e alla gestione della sessione.

A08:2021-Software and Data Integrity Failures

Sono problematiche legate a software importati da terze parti all'interno del proprio codice senza verificarne l'affidabilità e l'integrità dei dati.

A09:2021-Security Logging and Monitoring Failures

Consiste nell'insufficienza di strumenti di monitoraggio e controllo della sicurezza, la quale può portare a una mancata rilevazione delle minacce o a una reazione tardiva agli attacchi.

A10:2021-Server-Side Request Forgery

Consiste nel fatto che l'applicazione web apra una risorsa, il cui URL viene fornito dall'utente, senza prima validarlo.

In Figura 3.2 viene dunque illustrata una visione di insieme delle occorrenze in termini assoluti delle vulnerabilità appartenenti alle categorie dell'OWASP Top10-2021 in tutta la code base presa in considerazione, con l'ausilio di un grafico a barre:

3.4.3 Statistiche di ordine totale

Un'altra informazione rilevante riportata nella Tabella 3.1 è l'associazione tra la vulnerabilità e le applicazioni in cui questa è stata riscontrata almeno una volta. Si tenga presente che nelle colonne compaiono elencate solo le sette applicazioni a priorità alta e non compare la dicitura "others", stante a significare l'unione delle nove applicazioni a priorità media: questa decisione deriva dal fatto che in queste ultime nove applicazioni non si sono ricercate tutte le vulnerabilità presenti in tabella, ma si è voluto solo rafforzare il campione delle vulnerabilità presenti nella Top10 OWASP, e dunque negli output sono state appositamente filtrate solo questo tipo di vulnerabilità. Grazie all'associazione vulnerabilità-applicazioni, si possono inoltre tracciare dei grafici che delineano:

- Le 7 tipologie di vulnerabilità più presenti nella code base, ottenute dividendo le occorrenze di una certa vulnerabilità sul totale di tutte le vulnerabilità individuate, ed esprimendo il risultato in percentuale (Figura 3.3).
- Le 7 tipologie di vulnerabilità più diffuse nelle applicazioni (9, considerando i due casi di ex aequo finale), ottenendo dividendo il numero di applicazioni in cui la vulnerabilità è stata individuale per il totale delle applicazioni, ed esprimendo il risultato in percentuale (Figura 3.4).

Si può notare che in entrambi i grafici, i primi due posti sono occupati dalle stesse vulnerabilità:

- @RequestMapping without http method

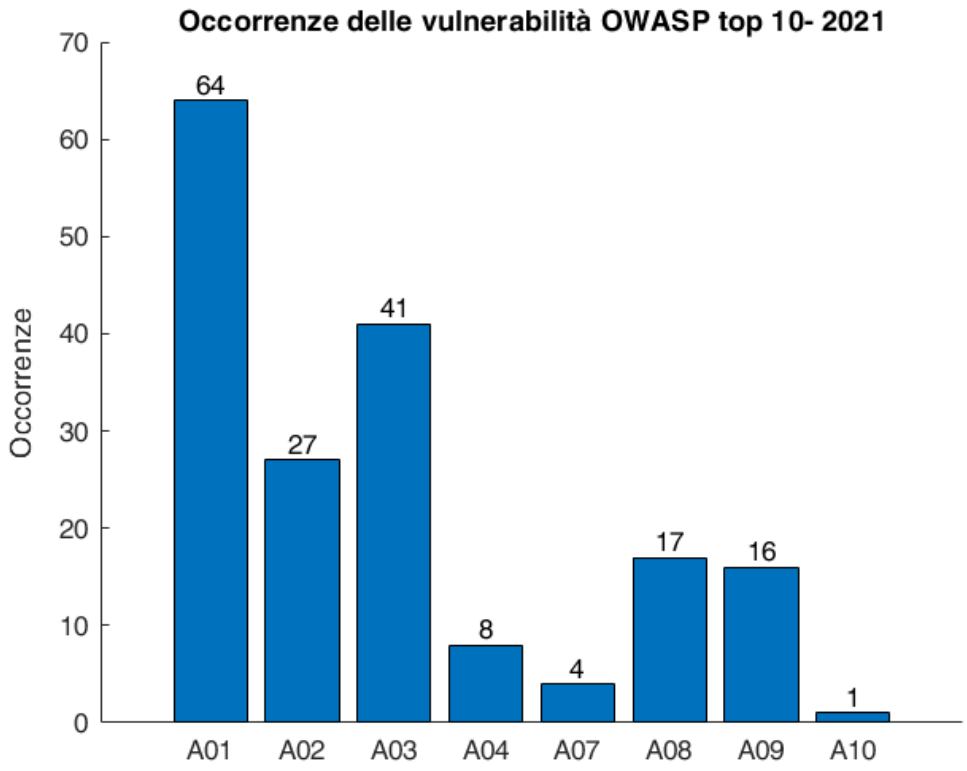

Figura 3.2: Occorrenze delle vulnerabilità appartenenti al ranking OWASP Top10 - 2021

- Not injected member of Spring

Nei due ranking sono inoltre presenti, sebbene in posizioni differenti, altre due vulnerabilità relative al framework Springboot:

- Optional accessed without isPresent
- Persistent Entity in @RequestMapping

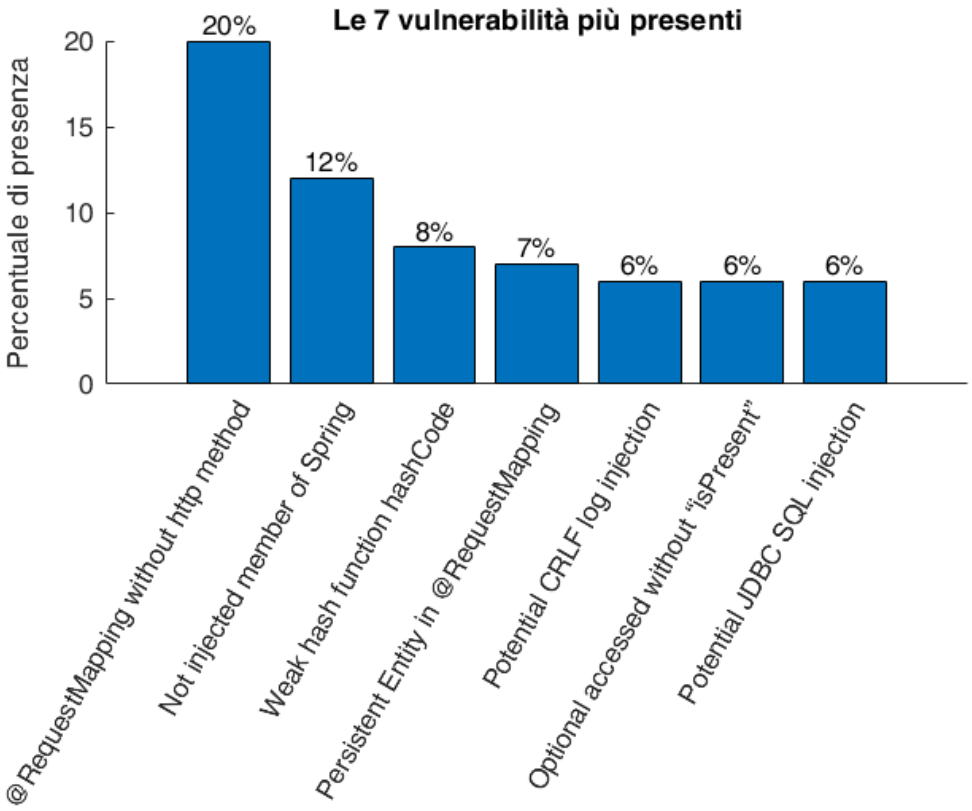

Figura 3.3: Le sette tipologie di vulnerabilità con più occorrenze nella code base, rispetto al totale delle occorrenze individuate

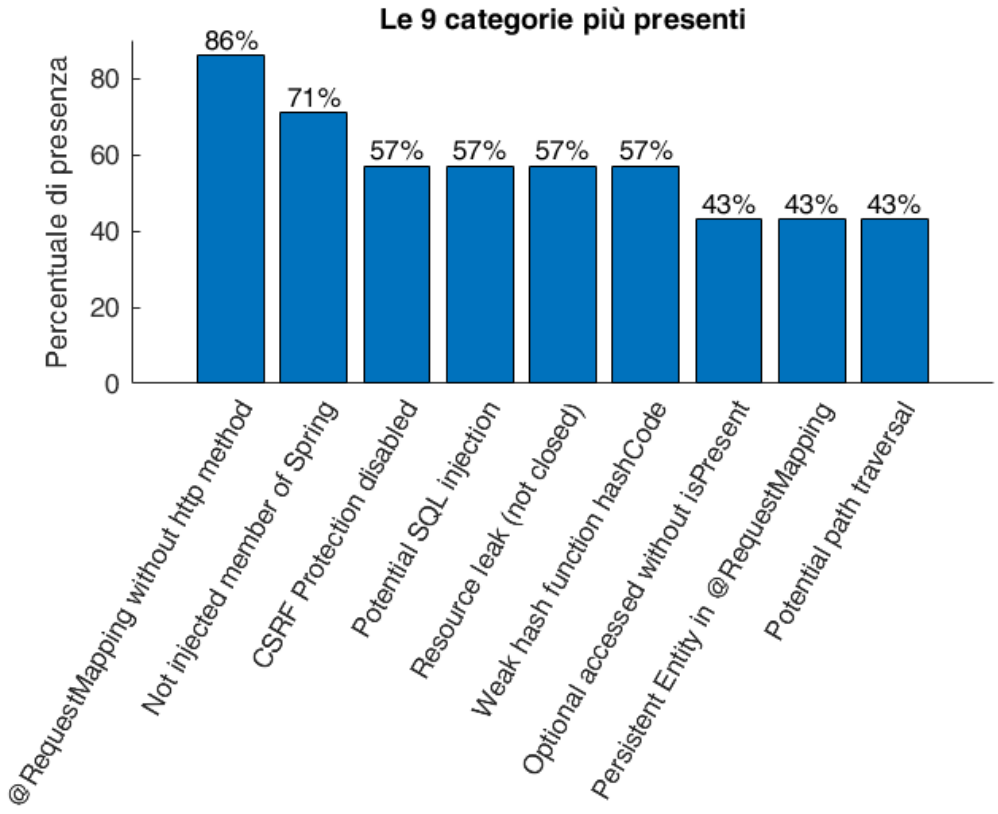

Figura 3.4: Le nove tipologie di vulnerabilità più presenti nelle applicazioni

3.5 Caso studio: OWASP VulnerableApp Project

Come discusso nel Paragrafo 2.2.1, in questo lavoro di tesi si è valutato di considerare esclusivamente applicazioni comuni, che non avessero finalità di testing per scopi di analisi statica della sicurezza. Un'eccezione è stata però compiuta in questo senso utilizzando l'applicazione VulnerableApp [26], progetto Open Source a cura della OWASP Foundation, e il cui codice è liberamente disponibile su Github.

Quest'applicazione si pone un obiettivo più ampio che quello di essere un semplice benchmark comprendente delle problematiche di sicurezza: è piuttosto un progetto didattico che mette in luce le caratteristiche di ciascuna vulnerabilità, fornendone più versioni; in questo modo, si cerca di far comprendere al discente le diverse gradazioni di pericolosità che può assumere un frammento di codice a seconda delle accortezze con cui lo si scrive. Inoltre, a differenza di altre applicazioni volutamente vulnerabili presenti sul mercato, VulnerableApp si presenta come un sistema scalabile, in cui gli sviluppatori possono aggiungere nuove versioni alle vulnerabilità esistenti, o anche aggiungere nuove categorie di vulnerabilità, per mezzo di template predefiniti.

3.5.1 Vulnerabilità riscontrate e loro occorrenze

Al momento della scrittura di questa Tesi, VulnerableApp include volutamente 9 tipologie di vulnerabilità, e considerando i vari esempi comprendenti più versioni di ciascuna vulnerabilità, il totale come da elenco contenuto nella documentazione ufficiale di VulnerableApp è di 35 occorrenze.
Gli strumenti in esame non sono stati tuttavia in grado di individuare tutte le tipologie di vulnerabilità (e di conseguenza tutte le loro occorrenze), ma solo alcune di esse; d'altra parte, nel corso del lavoro di analisi sono state individuate alcune vulnerabilità intrinseche al codice, non elencate nella documentazione e dunque non intenzionali. Nella Tabella 3.2 vengono confrontate le tipologie di vulnerabilità elencate nella documentazione con quelle effettivamente individuate.

Vulnerabilità	Presente in elenco	Individuata
Always true expression		x
Command Injection	x	x
File Upload Vulnerability	x	
Hard coded password		x
Impossible cast		x
JWT Vulnerability	x	
Open Redirect	x	
Optional accessed without 'isPresent'		x
Possible null pointer dereference		x
Potential JDBC SQL injection	x	x
Potential Path Traversal	x	x
Potential SSRF	x	x
@RequestMapping without http method		x
Resource leak (not closed)		x
Weak Hash function hashcode		x
Weak Random method		x
XXE Vulnerability	x	
XXS Vulnerability	x	

Tabella 3.2: Lista delle vulnerabilità elencate nella documentazione e/o individuate nelle analisi

I risultati elencati nella Tabella 3.2 vengono tradotti in forma grafica tramite dei digrammi a torta in Figura 3.5 e in Figura 3.6, al fine di fornire una visualizzazione di insieme più intuitiva e immediata delle proporzioni tra i vari sottoinsiemi sia di categorie di vulnerabilità che delle loro occorrenze.

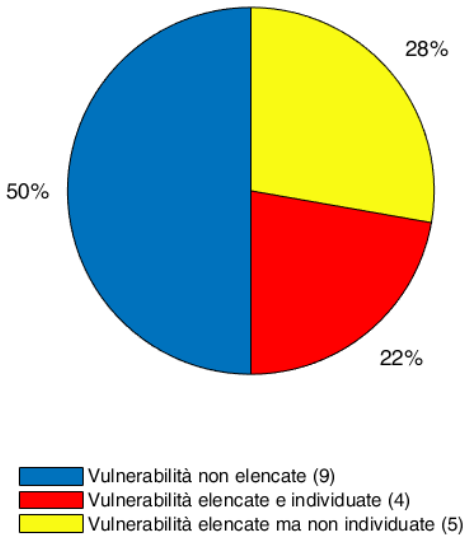

Figura 3.5: Distribuzione delle tipologie di vulnerabilità per VulnerableApp

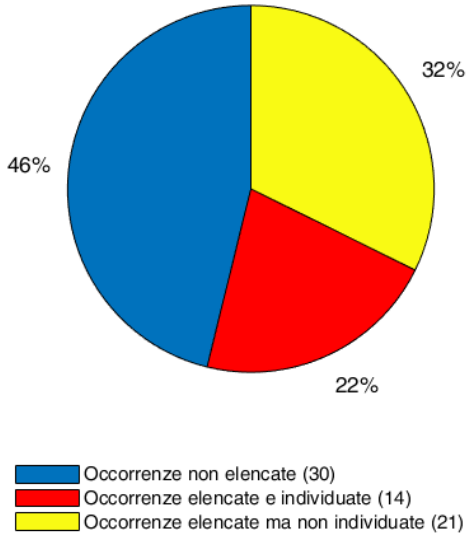

Figura 3.6: Distribuzione delle occorrenze delle vulnerabilità per VulnerableApp

Si può dunque notare come il 55% delle vulnerabilità in elenco (5 su 9) non vengano individuate dagli strumenti, percentuale che sale ancora al 60% (21 su 35) se ne si considerano le occorrenze: in altre parole, le vulnerabilità non individuate non sono dei casi isolati, ma al contrario costituiscono una possibilità niente affatto trascurabile.

Come accennato nel Paragrafo 2.1.1, conducendo un'analisi a posteriori si possono attribuire varie motivazioni alla base della mancata individuazione di questa significativa fetta di tipologie (e conseguentemente, di occorrenze di esse) da parte dei vari strumenti. In alcuni casi ciò è dipeso dall'utilizzo di licenze speciali degli strumenti,come quelle di prova o quelle Educational, chiaramente meno performanti rispetto alla rispettive versioni commerciali ad uso aziendale. In altri casi ciò è invece dipeso dalla scelta di utilizzare solo delle distribuzioni sotto forma di plugins: questa decisione, se da un lato contribuisce a creare uniformità di valutazione, dall'altra rende alcuni strumenti più limitati rispetto alle versioni integrate nelle pipeline CI/CD, come si può evincere dalle rispettive documentazioni.

Si potrebbe dunque ovviare a questa limitazione presentatasi in questo lavoro di tesi adottando strumenti di natura più professionale, o sperimentando le possibili variazioni in termini sia di tipologie di vulnerabilità che di quantità di loro occorrenze individuate utilizzando altre tipologie di integrazione.

3.5.2 Suddivisione dei risultati per strumento

Scendendo più nel dettaglio, si vogliono comparare tra loro le capacità di individuazione delle vulnerabilità dei vari strumenti: tale comparazione è disponibile in Figura 3.7 e in Figura 3.8, dove per mezzo di alcuni grafici a barre, si possono sia confrontare visivamente le performance degli strumenti, sia visualizzare graficamente le singole proporzioni per strumento.
Da notare che nei grafici compaiono solo sei dei sette strumenti adottati durante la fase di analisi: questo perché uno di essi, date probabilmente la particolare versione di integrazione e la licenza prese in considerazione, è stato in grado di individuare sensibilmente meno tipologie di vulnerabilità rispetto alla media degli altri strumenti ed è stato quindi successivamente scartato nella fase di report dei risultati.

Si può notare che le situazioni più critiche si riscontrino negli strumenti 2 e 5, non in grado di individuare neppure una delle vulnerabilità in elenco da VulnerableApp, mentre gli strumenti 1 e 4 risultano essere i più performanti. In questi due specifici casi degli strumenti 2 e 5, si può tuttavia affermare che gli scarsi risultati dipendano esclusivamente dalla versione di integrazione scelta, poiché gli strumenti in questione sarebbero effettivamente in grado di individuare le stesse quattro vulnerabilità degli strumenti 1 e 4 (oltre ad ulteriori due) se venissero utilizzati nella versione integrata nelle pipelines CI/CD.

Per quanto riguarda invece gli strumenti 3 e 6, che costituiscono situazioni mediocri individuando rispettivamente 3 e 2 delle vulnerabilità in elenco, si può affermare che invece ciò non dipenda tanto dalla versione di integrazione quanto dalla tipologia di licenze adottate, che risultano troppo limitate rispetto alle rispettive versioni commerciali, di livello di gran lunga più alto.

Si può notare come tuttavia tutti gli strumenti, seppure in modo variabile, presentino comunque una buona capacità di individuare le tipologie di vulnerabilità non in elenco e le rispettive occorrenze, segno che vulnerabilità considerabili di pericolosità minore sono presenti e ben individuate dagli strumenti anche all'interno di VulnerableApp.

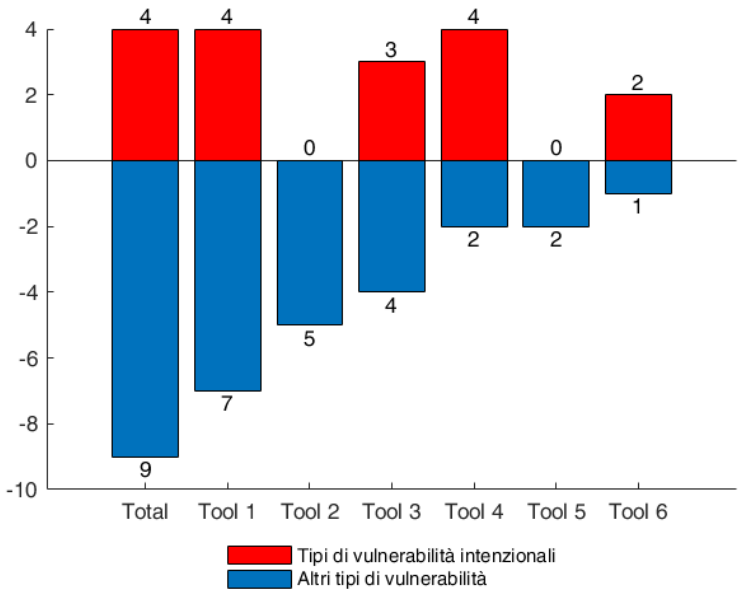

Figura 3.7: Distribuzione delle tipologie di vulnerabilità per VulnerableApp, suddivise per strumento

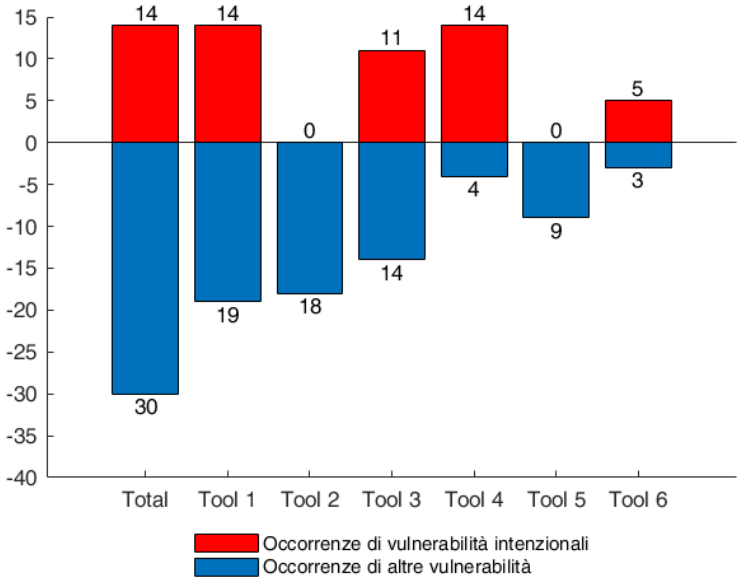

Figura 3.8: Distribuzione delle occorrenze di vulnerabilità per VulnerableApp, suddivise per strumento

Capitolo 4

Metodologia di valutazione degli strumenti

Questo capitolo e il Capitolo 5 verteranno sull'ultimo obiettivo di questo lavoro di tesi: la valutazione delle performance degli strumenti. I risultati delle analisi fin qui svolte verranno dunque ora rappresentati più ad alto livello e raggruppati in viste aggregate, in funzione della pericolosità e/o delle categorie di vulnerabilità individuate. Queste rappresentazioni di insieme torneranno utili ai fini del calcolo degli indici generali, indicatori valutativi delle prestazioni degli strumenti la cui natura sarà approfondita nel corso di questo capitolo. Questi ultimi indici verranno a loro volta combinati in un unico parametro globale, un valore quantitativo inteso come misura della performance: ciò consente non solo di avere una stima della bontà del singolo strumento, ma anche di confrontare i vari strumenti in una scala di misurazione comune. L'assegnazione del punteggio finale verrà effettuata seguendo la metodologia di scoring proposta dall' OWASP Foundation [4], che verrà altrettanto approfondita e discussa in questa sede.

Quasi tutti i conteggi che saranno oggetto di questo Capitolo e del Capitolo 5 sono stati effettuati tenendo conto delle sette applicazioni a priorità alta, mentre solo in alcuni specifici casi, che verranno appositamente evidenziati, essi sono stati integrati anche con i risultati provenienti dalle nove applicazioni a priorità media. Inoltre, come precedentemente accennato nella Sezione 3.5, verranno presi in considerazione solo sei dei sette strumenti selezionati, poiché perché uno di essi, essendo capace di individuare sensibilmente meno tipologie di vulnerabilità rispetto agli altri strumenti, è stato incluso nella fase di analisi ma verrà escluso in questo contesto di report dei risultati.

4.1 Realizzazione delle viste aggregate

Il modello valutativo che viene adottato in questo lavoro di tesi si basa, come vedremo nella Sezione 4.2, sui tassi di vera e falsa positività, che a loro volta si basano sui valori dei flags di tipo TP, FP, TN e FN, per la cui attribuzione si rimanda al Capitolo 3.3. Ai fini del calcolo dei tassi di positività e negatività è necessario sommare per flags distinti le etichette che sono state assegnate a un certo insieme di vulnerabilità: in altre parole, è opportuno creare delle viste aggregate, in cui tutte le vulnerabilità individuate vengano raggruppate in base ad un qualche criterio, per ottenere un risultato di somma delle quattro diverse tipologie di flags. Nei seguenti paragrafi verranno illustrate tutte le viste aggregate su cui si è operato.

4.1.1 Vista totale

Si tratta di un caso particolare in cui non c'è stato nessun raggruppamento, e tutte le vulnerabilità individuate (relative in questo caso solo alle sette applicazioni a priorità alta) sono state considerate come facenti parte di un'unica grande applicazione; questa vista è dunque molto utile ad avere una visione di insieme su quante vulnerabilità siano presenti all'interno di tutta la code base. Sono state inoltre create due sotto-categorie, comprendenti rispettivamente tutte le vulnerabilità di tutte le applicazioni che fossero classificate come di pericolosità alta, e tutte quelle di pericolosità bassa: si sono quindi ottenuti 3 livelli di aggregazione totali, per un totale di $3(livelli) \times 6(strumenti) \times 4(flags) = 72$ valori di somma.

Nelle Tabelle 4.1,4.2 e 4.3 sono riportati i risultati ottenuti considerando rispettivamente tutte le vulnerabilità, solo quelle a pericolosità alta e solo quelle a pericolosità bassa.

	FN_TOT	**FP_TOT**	**TN_TOT**	**TP_TOT**	**TOT**
STRUMENTO 1	14	12	22	113	**161**
STRUMENTO 2	11	13	6	73	**103**
STRUMENTO 3	12	15	12	90	**129**
STRUMENTO 4	19	2	8	27	**56**
STRUMENTO 5	12	2	3	11	**28**
STRUMENTO 6	6	5	5	24	**40**
TOT	**74**	**49**	**56**	**338**	**517**

Tabella 4.1: Conteggi globali relativi alla vista totale

	FN_HIGH	FP_HIGH	TN_HIGH	TP_HIGH	TOT
STRUMENTO 1	7	12	13	104	**136**
STRUMENTO 2	11	12	6	73	**102**
STRUMENTO 3	12	15	11	42	**80**
STRUMENTO 4	16	0	8	25	**49**
STRUMENTO 5	12	2	3	9	**26**
STRUMENTO 6	6	5	2	17	**30**
TOT	**64**	**46**	**43**	**270**	**423**

Tabella 4.2: Conteggi relativi alla vista totale per le vulnerabilità a pericolosità alta

	FN_LOW	FP_LOW	TN_LOW	TP_LOW	TOT
STRUMENTO 1	7	0	9	9	**25**
STRUMENTO 2	0	1	0	0	**1**
STRUMENTO 3	0	0	1	48	**49**
STRUMENTO 4	3	2	0	2	**7**
STRUMENTO 5	0	0	0	2	**2**
STRUMENTO 6	0	0	3	7	**10**
TOT	**10**	**3**	**13**	**68**	**94**

Tabella 4.3: Conteggi relativi alla vista totale per le vulnerabilità a pericolosità bassa

4.1.2 Vista per applicazione

Si tratta dell'aggregazione più intuitiva, in cui le vulnerabilità raccolte sono state raggruppate per applicazione (anche in questo caso, considerando solo le sette applicazioni a priorità alta). In questo modo, per ognuno dei sei strumenti, è stato possibile calcolare il totale delle occorrenze di flags TP,FP,TN e FN, ottenendo un totale di $7(applicazioni) \times 6(strumenti) \times 4(flags) = 168$ valori di somma. Questa vista è perciò utile a comprendere quali applicazioni siano le più soggette a rischi di sicurezza informatica, e a correlare la maggior presenza di vulnerabilità con il maggior numero di linee di codice dell'applicazione.

Le seguenti Tabelle mostrano i risultati ottenuti, suddivisi per applicazione: analizzando ad esempio la Tabella 4.4, relativa all'app "cwa-server", si può notare come per lo strumento 1 i valori TP,FP,TN e FN siano rispettivamente pari a 5, 1, 5 e 3.

	FN	**FP**	**TN**	**TP**	**TOT**
STRUMENTO 1	3	1	5	5	**14**
STRUMENTO 2	0	1	1	9	**11**
STRUMENTO 3	2	3	1	12	**18**
STRUMENTO 4	1	0	1	1	**3**
STRUMENTO 5	0	0	0	0	**0**
STRUMENTO 6	1	1	0	2	**4**
TOT	**7**	**6**	**8**	**29**	**50**

Tabella 4.4: Conteggi relativi all'applicazione "cwa-server"

	FN	**FP**	**TN**	**TP**	**TOT**
STRUMENTO 1	2	7	5	3	**17**
STRUMENTO 2	3	3	4	3	**13**
STRUMENTO 3	0	1	8	2	**11**
STRUMENTO 4	0	0	7	1	**8**
STRUMENTO 5	0	0	0	2	**2**
STRUMENTO 6	0	4	4	7	**15**
TOT	**5**	**15**	**28**	**18**	**66**

Tabella 4.5: Conteggi relativi all'applicazione "dm-store"

	FN	**FP**	**TN**	**TP**	**TOT**
STRUMENTO 1	0	1	7	7	**15**
STRUMENTO 2	0	4	0	14	**18**
STRUMENTO 3	0	5	3	1	**9**
STRUMENTO 4	3	0	0	5	**8**
STRUMENTO 5	2	1	0	0	**3**
STRUMENTO 6	0	0	0	0	**0**
TOT	**5**	**11**	**10**	**27**	**53**

Tabella 4.6: Conteggi relativi all'applicazione "employee management"

	FN	**FP**	**TN**	**TP**	**TOT**
STRUMENTO 1	1	0	1	10	**12**
STRUMENTO 2	1	1	0	9	**11**
STRUMENTO 3	0	1	0	8	**9**
STRUMENTO 4	9	0	0	2	**11**
STRUMENTO 5	0	0	0	0	**0**
STRUMENTO 6	0	0	1	3	**4**
TOT	**11**	**2**	**2**	**32**	**47**

Tabella 4.7: Conteggi relativi all'applicazione "jcart"

	FN	**FP**	**TN**	**TP**	**TOT**
STRUMENTO 1	2	3	3	36	**44**
STRUMENTO 2	0	1	1	13	**15**
STRUMENTO 3	0	4	0	20	**24**
STRUMENTO 4	0	2	0	1	**3**
STRUMENTO 5	0	1	0	3	**4**
STRUMENTO 6	0	0	0	0	**0**
TOT	**2**	**11**	**4**	**73**	**90**

Tabella 4.8: Conteggi relativi all'applicazione "Poplar"

	FN	**FP**	**TN**	**TP**	**TOT**
STRUMENTO 1	4	0	0	29	**33**
STRUMENTO 2	7	0	0	11	**18**
STRUMENTO 3	10	0	0	15	**25**
STRUMENTO 4	5	0	0	13	**18**
STRUMENTO 5	8	0	0	1	**9**
STRUMENTO 6	5	0	0	3	**8**
TOT	**39**	**0**	**0**	**72**	**111**

Tabella 4.9: Conteggi relativi all'applicazione "Vulnerable App"

	FN	**FP**	**TN**	**TP**	**TOT**
STRUMENTO 1	2	0	1	23	**26**
STRUMENTO 2	0	3	0	14	**17**
STRUMENTO 3	0	1	0	32	**33**
STRUMENTO 4	1	0	0	4	**5**
STRUMENTO 5	2	0	3	5	**10**
STRUMENTO 6	0	0	0	9	**9**
TOT	**5**	**4**	**4**	**87**	**100**

Tabella 4.10: Conteggi relativi all'applicazione "yugastore"

4.1.3 Vista per categoria

In questo tipo di vista, tutte le segnalazioni raccolte nel corso del processo di analisi sono state raggruppate in base a sedici specifiche categorie di vulnerabilità, che vengono elencate nelle successive Tabelle 4.11 e 4.12.
Di queste sedici categorie, otto sono semplici, ossia categorie che corrispondono ad un unico tipo di vulnerabilità; le seguenti otto sono invece categorie OWASP, all'interno di ognuna delle quali si possono trovare anche più etichette vulnerabilità differenti. Le categorie OWASP sono in realtà 10, come già discusso nel Paragrafo 3.4.2: la presenza di sole otto categorie deriva dal fatto che nella code base oggetto di questo lavoro di tesi non sono mai state riscontrate vulnerabilità di tipologia "A05- Security Misconfiguration" e "A06-Vulnerable and Outdated Components".

C'è da sottolineare un'importante differenza: nelle categorie OWASP (tratte dalla Top10 OWASP 2021), sono state considerate sia le vulnerabilità relative alle sette applicazioni a priorità alta che quelle relative alle nove applicazioni a priorità media, mentre nelle categorie semplici si sono considerate solo le applicazioni a priorità alta. Questa distinzione è dovuta al fatto che metodologia OWASP è stata parte fortemente integrante di questo lavoro di tesi, e dunque si è scelto di approfondire maggiormente le dieci categorie ad essa attinenti, attingendo all'ulteriore fonte di dati delle applicazioni a priorità media per poter ottenere dei risultati statisticamente più significativi.

Il totale risulta dunque di $16(categorie) \times 6(strumenti) \times 4(flags) = 384$ valori di somma. Questa vista permette di analizzare singolarmente sia le categorie di vulnerabilità più pericolose (come quelle relative all'OWASP Top 10), sia di poter condurre altre statistiche su vulnerabilità non particolarmente pericolose ma molto frequenti (come la categoria riguardante tutte le vulnerabilità relative al framework Spring).

Si riportano nelle Tabelle 4.11 e 4.12 i valori ottenuti per le vulnerabilità rispettivamente appartenenti alle categorie semplici e a quelle OWASP, suddivisi per categorie (sulle righe) e strumenti (sulle colonne).

		S1	S2	S3	S4	S5	S6
Command Injection	**TP**	2	0	0	2	0	0
Command Injection	**FP**	0	0	0	0	0	0
Command Injection	**TN**	0	0	0	0	0	0
Command Injection	**FN**	0	0	2	0	0	2
LDAP injection	**TP**	0	0	1	0	0	0
LDAP injection	**FP**	0	0	0	0	0	0
LDAP injection	**TN**	0	0	0	0	0	0
LDAP injection	**FN**	1	0	0	0	0	0
Path Traversal	**TP**	2	0	0	3	0	0
Path Traversal	**FP**	2	0	0	0	0	0
Path Traversal	**TN**	0	0	2	2	0	2
Path Traversal	**FN**	1	0	3	0	0	3
SQL injection	**TP**	8	0	3	6	0	0
SQL injection	**FP**	6	0	7	0	0	0
SQL injection	**TN**	7	0	6	6	0	0
SQL injection	**FN**	0	0	5	2	0	0
Weak Hash	**TP**	2	0	32	2	0	0
Weak Hash	**FP**	2	0	2	2	0	0
Weak Hash	**TN**	0	0	0	0	0	0
Weak Hash	**FN**	0	0	0	0	0	0
Weak Random	**TP**	1	0	1	0	0	0
Weak Random	**FP**	0	2	0	0	0	0
Weak Random	**TN**	2	0	2	0	0	0
Weak Random	**FN**	0	0	0	0	0	0
Spring related	**TP**	56	48	48	12	0	15
Spring related	**FP**	1	5	0	0	0	0
Spring related	**TN**	0	0	2	0	0	3
Spring related	**FN**	0	0	0	13	0	0
Other vulnerabilities	**TP**	21	30	2	0	10	1
Other vulnerabilities	**FP**	1	6	2	0	2	0
Other vulnerabilities	**TN**	5	1	0	0	3	0
Other vulnerabilities	**FN**	6	4	2	8	12	0

Tabella 4.11: Conteggi relativi alle 8 categorie semplici

		S1	S2	S3	S4	S5	S6
A01- Broken Access Control	**TP**	58	0	48	11	0	8
A01- Broken Access Control	**FP**	3	0	1	0	0	0
A01- Broken Access Control	**TN**	0	0	3	2	0	2
A01- Broken Access Control	**FN**	1	0	3	0	0	3
A02- Cryptographic failures	**TP**	3	0	33	2	0	0
A02- Cryptographic failures	**FP**	2	4	2	2	0	0
A02- Cryptographic failures	**TN**	4	0	4	0	0	0
A02- Cryptographic failures	**FN**	0	0	0	0	0	0
A03- Injection	**TP**	26	1	4	8	0	0
A03- Injection	**FP**	6	0	7	0	0	0
A03- Injection	**TN**	7	0	7	6	0	0
A03- Injection	**FN**	1	0	7	2	0	2
A04- Insecure Design	**TP**	7	0	1	1	0	0
A04- Insecure Design	**FP**	0	0	1	0	0	0
A04- Insecure Design	**TN**	1	0	0	0	0	1
A04- Insecure Design	**FN**	0	0	0	0	0	0
A07- Identification/Authentication failures	**TP**	1	0	2	1	0	0
A07- Identification/Authentication failures	**FP**	0	0	1	0	0	0
A07- Identification/Authentication failures	**TN**	1	0	0	0	0	0
A07- Identification/Authentication failures	**FN**	1	0	0	0	0	0
A08- SW and data integrity failures	**TP**	0	17	0	4	0	0
A08- SW and data integrity failures	**FP**	0	0	0	0	0	0
A08- SW and data integrity failures	**TN**	0	0	0	0	0	0
A08- SW and data integrity failures	**FN**	0	0	0	13	0	0
A09- Security logging/monitoring failures	**TP**	16	0	0	0	0	0
A09- Security logging/monitoring failures	**FP**	0	0	0	0	0	0
A09- Security logging/monitoring failures	**TN**	0	0	0	0	0	0
A09- Security logging/monitoring failures	**FN**	0	0	0	0	0	0
A10-SSRF	**TP**	1	0	0	1	0	0
A10-SSRF	**FP**	0	0	0	0	0	0
A10-SSRF	**TN**	0	0	0	0	0	0
A10-SSRF	**FN**	0	0	0	0	0	0

Tabella 4.12: Conteggi relativi alle 8 categorie OWASP

4.1.4 Vista totale OWASP

In questo tipo di vista sono state riunite tutte le segnalazioni di vulnerabilità facenti parte delle otto categorie OWASP di cui al Paragrafo 4.1.3. Questa vista permette dunque di avere un quadro generale della situazione riguardo la presenza nel codice di vulnerabilità particolarmente pericolose, su cui è fortemente necessario intervenire al più presto. Si sono dunque calcolati in totale $6(strumenti) \times 4(flags) = 24$ valori di somma .

Viene qui riportata la tabella finale dei risultati:

	FN_TOT	**FP_TOT**	**TN_TOT**	**TP_TOT**	**TOT**
STRUMENTO 1	4	11	15	182	**212**
STRUMENTO 2	0	4	0	20	**24**
STRUMENTO 3	13	16	14	91	**134**
STRUMENTO 4	17	2	8	43	**70**
STRUMENTO 5	0	0	0	0	**0**
STRUMENTO 6	6	0	3	20	**29**
TOT	**40**	**33**	**40**	**356**	**469**

Tabella 4.13: Conteggi relativi alla vista totale OWASP

4.2 Metodologia di scoring

In questa Sezione verrà descritto il modello teorico di attribuzione del punteggio alle prestazioni degli strumento partendo dal calcolo degli indici valutativi principali, sulla cui base si potrà costruire un indice di valutazione unico seguendo la metodologia proposta dall'OWASP Foundation [4].

4.2.1 Calcolo degli indici valutativi

Partendo dalle sommatorie dei flags TP,TN,FP e FN, ottenute come descritto nella Sezione 4.1, è possibile calcolare alcuni importanti indici valutativi delle prestazioni degli strumenti, che torneranno utili nella successiva determinazione del parametro globale di scoring: in questa sede i due indici di principale interesse sono stati il TPR e il FPR.

True Positive Rate

Il tasso di veri positivi(in inglese TPR, True Positive Rate) si calcola come:

$$TPR = \frac{TP}{TP + FN}$$

Esso è dunque una misura di quante vere vulnerabilità siano state individuate dallo strumento, rispetto al totale di tutte le vere vulnerabilità presenti nel codice (siano esse di tipo true positive o false negative). E' anche detto sensibilità o recall, ed è un indice che si estende nel dominio compreso tra 0 e 1. Crescendo al crescere dell'indice di vera positività, e decrescendo al decrescere dell'indice di falsa negatività, è proporzionale alle prestazioni dello strumento.

False Positive Rate

Analogamente, il tasso di falsi positivi(in inglese FPR, False Positive Rate) si calcola come:

$$FPR = \frac{FP}{FP + TN}$$

Esso è dunque una misura di quante segnalazioni che si siano rivelate come dei falsi positivi lo strumento abbia individuato, rispetto al totale di tutte le non-vulnerabilità presenti nel codice. E' anche detto fall-out, e considerando che esso cresce al crescere del numero di falsi positivi e decresce al decrescere del numero di veri negativi, si può considerare un indice inversamente proporzionale alle prestazioni dello strumento, anch'esso compreso nel dominio tra 0 e 1.

Precisione e Accuratezza

I due tassi sopra descritti sono in qualche maniera legati, ma non devono essere confusi, con i concetti di precisione e accuratezza dello strumento.
La precisione si calcola come

$$PPV = \frac{TP}{TP + FP}$$

Essa si può dunque intendere come il tasso di vulnerabilità *vere* che lo strumento ha individuato, rispetto a tutte quelle che ha individuato (che comprendono anche le segnalazioni di falsi positivi); essa è dunque, in maniera complementare, una misura di quanto lo strumento sia stato in grado di evitare di riportare segnalazioni di falsi positivi. E' anche indicata come PPV, che sta per positive predictive value.

L'accuratezza si può invece esprimere come:

$$ACC = \frac{TP + TN}{TP + TN + FP + FN}$$

Essa è quindi un indice simile alla precisione, ma che tiene conto anche delle non-vulnerabilità, essendo infatti interpretabile come il numero di segnalazioni corrette (siano esse positive o negative) rispetto al totale delle segnalazioni; in altre parole,l'accuratezza è una misura di quanto lo strumento sia stato in grado di evitare report errati, sia in eccesso (casi di false positive) che in difetto (casi di false negative) rispetto al dovuto.

4.2.2 Metodologia di scoring proposta dall'OWASP Foundation

Partendo dai due indici di TPR e FPR è possibile calcolare un unico parametro di misurazione delle prestazioni degli strumenti, seguendo la metodologia OWASP presentata all'interno dell'OWASP Benchmark project [4], un progetto che mira a fornire dei benchmark utili sia per test di SAST che di DAST Analaysis. L'attribuzione del punteggio unico è visualizzabile graficamente attraverso la cosiddetta OWASP Scorecard, tavola valutativa della cui versione ufficiale viene presentato in Figura 4.1 un riadattamento grafico, creato ai fini di questo lavoro di tesi tramite uno script Matlab.

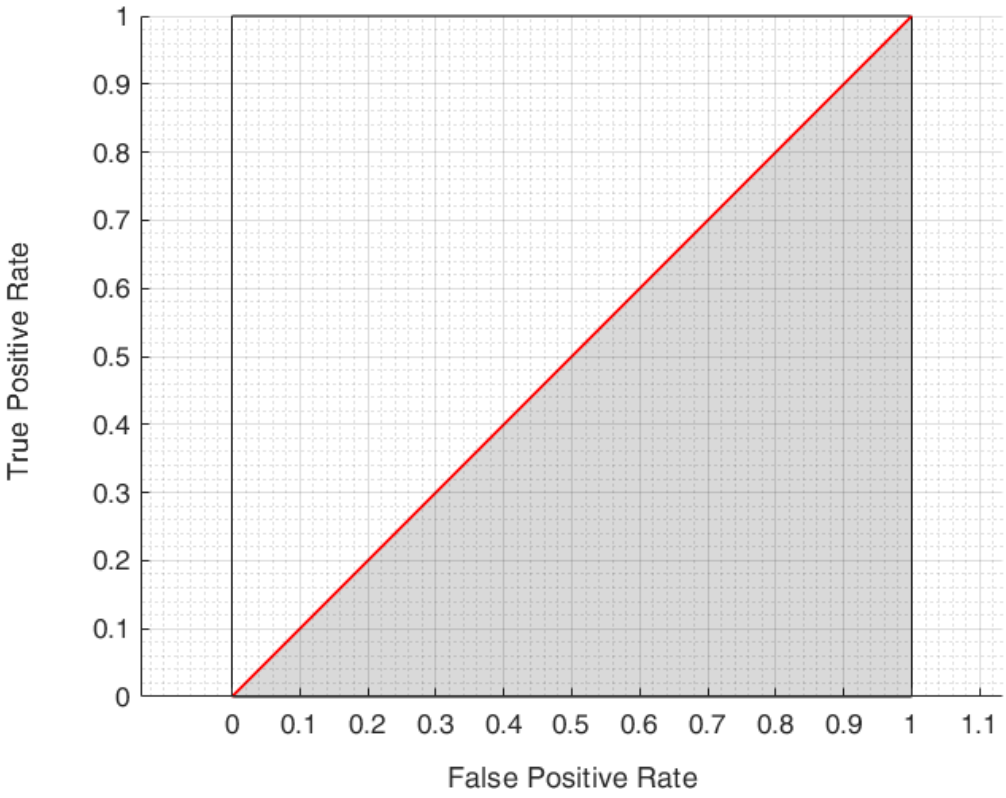

Figura 4.1: Riproduzione della Scorecard ufficiale dell'OWASP Benchmark project

In questo grafico vengono riportati sull'asse delle ascisse il valore del False Positive Rate e su quello delle ordinate il True Positive Rate, in forma percentuale (o, equivalentemente, in un range compreso tra 0 e 1). Come descritto nel Paragrafo precedente, il TPR è proporzionale alla bontà dello strumento, mentre il FPR ne è inversamente proporzionale: ne consegue che un ipotetico strumento ideale, che

riuscisse a individuare tutte e sole le vere vulnerabilità presente nel codice (e dunque non riportasse mai false segnalazioni come FP o FN), si troverebbe nell'angolo in alto a sinistra della Scorecard; viceversa, un ipotetico strumento che non riuscisse a individuare neppure una vera vulnerabilità, ma al contrario segnalasse solo casi di falsa positività, si troverebbe nell'angolo in basso a destra. Uno strumento che non riportasse nulla, non riporterebbe falsi positivi, ma neppure veri positivi, e dunque si collocherebbe nell'angolo in basso a sinistra; infine uno strumento che riportasse tutte le segnalazioni possibili, riporterebbe tutti i veri positivi, ma anche tutti i falsi positivi e dunque si collocherebbe in alto a destra. Chiaramente, si tratta di casi estremi, poiché la maggior parte degli strumenti reali si colloca all'interno del quadrato.

La diagonale in rosso è detta di "random guessing": uno strumento che si trovi su questa retta presenta TPR e FPR coincidenti ed è dunque perfettamente equivalente, dal punto di vista probabilistico, ad un oracolo casuale che assegna in maniera randomica i flags TP, TN, FP ed FN alle vulnerabilità. Ad uno strumento di questo tipo verrà chiaramente assegnato un punteggio di prestazioni pari a 0, mentre ne consegue che uno strumento sopra la diagonale (area bianca) avrà un punteggio maggiore di zero e uno sotto la diagonale (area grigia) avrà punteggio minore di zero.

In particolare, il parametro valutativo globale consiste nella distanza punto-retta tra il punto P (avente come coordinate il FPR e il TPR dello strumento) e la diagonale del quadrato, distanza che sarà chiaramente perpendicolare alla diagonale stessa. In formule, si ottiene che il parametro valutativo unico d si può esprimere come:

$$d = \frac{TPR - FPR}{\sqrt{2}}$$

La distanza d è dunque un indice che si estende nel dominio compreso tra $\frac{-1}{\sqrt{2}}$ e $\frac{1}{\sqrt{2}}$. In Figura 4.2 viene riportato un esempio in cui sono stati inseriti, a puro scopo illustrativo, dei dati *casuali* di true positive rate e false positive rate per i sei strumenti: si può notare come d, indicata fra parentesi in legenda, possa assumere sia valori positivi che negativi a seconda della posizione, rispettivamente sopra o sotto la diagonale principale, ed aumenti in valore assoluto al crescere della lontananza da essa.

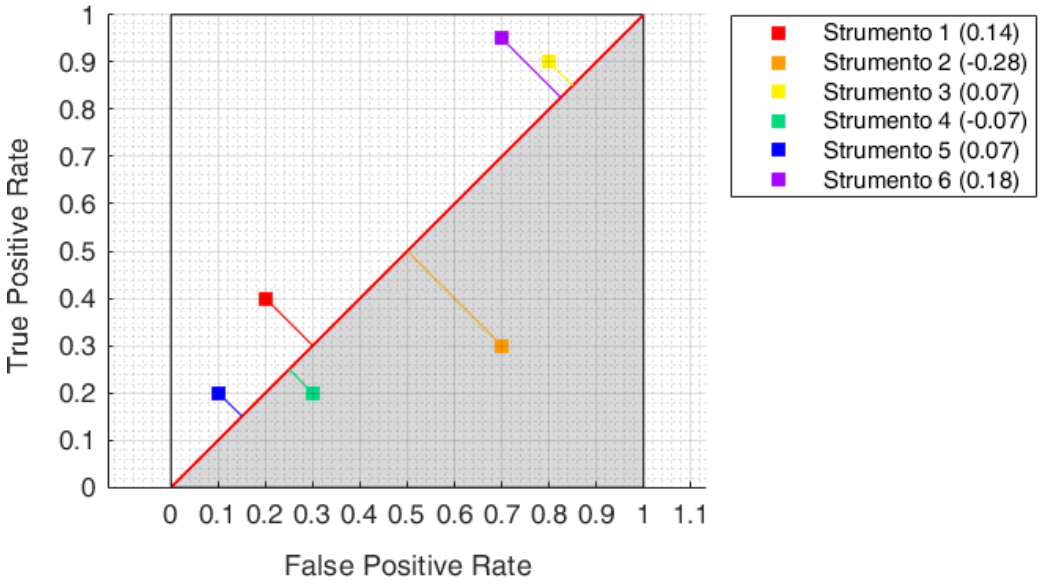

Figura 4.2: Riadattamento grafico dell'OWASP Scorecard, con inserimento di dati casuali a scopo illustrativo

4.2.3 Analogie con l'indice di Youden

Il modello teorico di attribuzione del punteggio proposto dalla OWASP Foundation e descritto nel Paragrafo 4.2.2 non è l'unico possibile ai fini del calcolo delle prestazioni: esistono infatti diversi indici utili a questo scopo, tra cui si possono nominare l'indice di Youden, l'F-measure, il coefficiente di correlazione di Matthews. L'obiettivo di questo Paragrafo è quello di approfondire esclusivamente l'indice di Youden, in quanto trattasi di un coefficiente strettamente correlato al modello teorico OWASP adottato nel corso di questo lavoro di Tesi.
L'indice di Youden è un indice valutativo proposto da W.J. Youden nel 1950, secondo cui la misura della performance J si può esprimere come:

$$J = TPR + TNR - 1 = \frac{TP}{TP + FN} + \frac{TN}{TN + FP} - 1$$

Per TPR si intende il True Positive Rate, o sensibilità ,di cui si è già discusso precedentemente e che si è detto essere proporzionale alle prestazioni dello strumento. Con TNR si intende invece il True Negative Rate, o specificità, ossia una misura di quante non-vulnerabilità siano state effettivamente categorizzate dallo strumento come dei veri negativi: si tratta dunque, in modo complementare, di una misura di quante segnalazioni di falsi positivi lo strumento sia stato in grado di evitare, e ne consegue che sia un indice proporzionale alle sue prestazioni.

Entrambi i rate si estendono nel dominio tra 0 e 1, e lo stesso vale per J; in particolare, l'indice di Youden si può interpretare come una misura del grado di decisionalità o di randomicità delle decisioni dello strumento: se J=0, si può affermare che lo strumento sia equiparabile ad un assegnatore randomico di flags, mentre il caso J=1 identifica lo strumento ideale, che compie solo decisioni perfette non riportando mai segnalazioni false, sia in positivo che in negativo. Anche per l'indice di Youden si può costruire un grafico simile alla Scorecard OWASP, che viene illustrato in Figura 4.3: in questo caso, sull'asse delle ascisse si trova il TNR, mentre su quello delle ordinate il TPR, e la misura J è la distanza dalla diagonale del quadrato, misurata però in maniera verticale.

Si può intravedere un parallelo con la misura d, utilizzata in questo lavoro di Tesi e pari alla distanza del punto (FPR;TPR) dalla diagonale del quadrato di lato 1: in effetti si ha che

$$TNR - 1 = \frac{TN}{TN + FP} - 1 = \frac{TN - TN - FP}{TN + FP} = \frac{-FP}{TN + FP} = -FPR$$

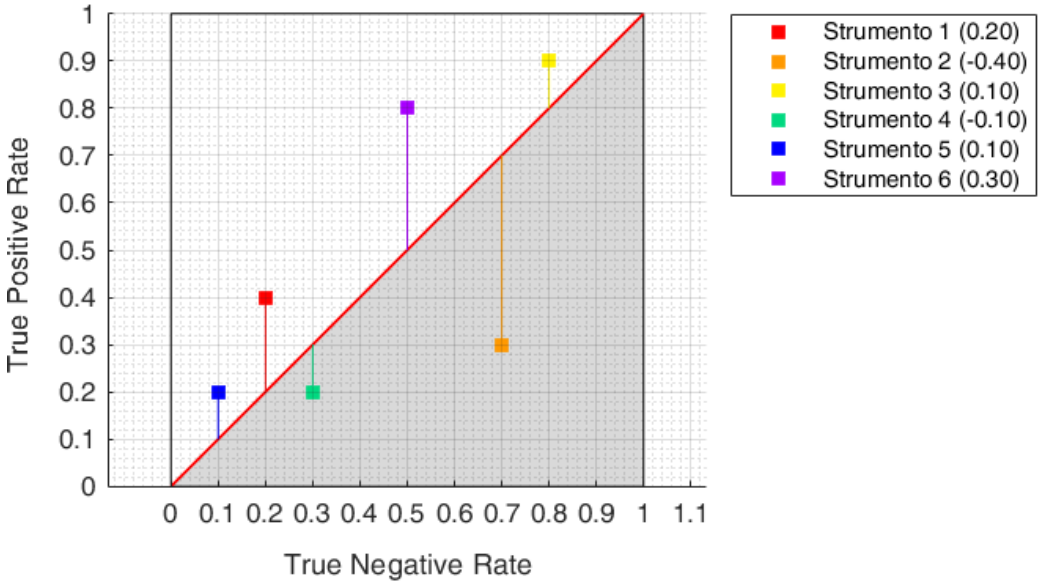

Figura 4.3: Grafico equivalente della Scorecard nel caso dell'indice di Youden

e dunque

$$J = TPR + TNR - 1 = TPR - FPR$$

La distanza d è stata invece calcolata come :

$$d = \frac{TPR - FPR}{\sqrt{2}}$$

Dunque si può affermare che d non sia altro che una versione riscalata tra $\frac{-1}{\sqrt{2}}$ e $\frac{1}{\sqrt{2}}$ dell'indice di Youden, che si estende tra 0 e 1. D'altronde se J=0, significa che TPR-FPR=0, ossia che TPR=FPR, retta che identifica il luogo dei punti sulla diagonale principale del quadrato, che si è discusso essere la linea di random guessing; dire invece che J=1, è equivalente a dire che TPP-FPR=1, ossia TPR=FPR+1, che ha come unica soluzione il punto in alto a sinistra del quadrato OWASP, in cui si trova lo strumento ideale.

Capitolo 5

Risultati ottenuti

In questo Capitolo si passerà ad elencare e discutere i risultati numerici finali, ottenuti grazie al raggruppamento in viste analizzato nella Sezione 4.1 e all' applicazione pratica della metodologia OWASP descritta in maniera teorica nella Sezione 4.2.
Per quanto riguarda l'organizzazione, verrà ripresa la suddivisione adottata nella Sezione 4.1, al fine di far corrispondere le categorie dei risultati finali alle viste già delineate in precedenza.

5.1 Limitazioni della ricerca

Prima di procedere alla presentazione dei risultati, risulta doveroso evidenziare alcune considerazioni sui limiti della ricerca condotta in questo lavoro di Tesi, per una corretta interpretazione di quanto contenuti nei seguenti paragrafi.

Ampiezza del campione

Il campione qui considerato, seppur di dimensioni totali considerevoli (si veda il Paragrafo 3.4.1), potrebbe non essere numericamente significativo ai fini statistici in alcune specifiche sotto-casistiche, come ad esempio quelle relative alle vulnerabilità di pericolosità bassa o quelle relative a specifiche categorie di vulnerabilità. Per questa ragione, si sono saltuariamente verificati dei casi in cui il TPR e/o il FPR hanno assunto valori pari a 0 o 1 mentre, come discusso nel Paragrafo 4.2.2, questi valori dovrebbero invece essere caratteristici soltanto di casi estremi.

Bias di identificazione dei casi negative

I risultati risentono del bias di identificazione sui casi "negative" presentato nel Paragrafo 3.3.2: essi non corrispondono infatti a tutte le vere e false non-vulnerabilità

del codice rilevate dallo strumento in sé, ma sono ottenuti mediante confronto diretto tra i vari strumenti. In altre parole, se ad esempio neppure uno strumento ha individuato una certa vulnerabilità, questa non potrà mai essere inserita nel conteggio dei falsi negativi di nessun altro strumento; allo stesso modo, il conteggio dei veri negativi è effettuato in base al fatto che almeno uno strumento abbia individuato una certa vulnerabilità in modo errato, e solo in questo modo essa viene aggiunta al conteggio dei TN di tutti gli altri che non l'abbiano riscontrata. Questa dipendenza dei casi negative dal confronto degli strumenti si riscontra in fase di risultati, constatando come i valori di TN e/o FN siano talvolta nulli.

Combinazione delle precedenti

La combinazione dei due fattori precedenti può sfociare in un risultato indeterminato: ad esempio, nei casi in cui il flag positive (sia esso true o false) sia nullo poiché ci si trova davanti a un campione non numericamente significativo, e contemporaneamente anche il flag negative sia nullo (per la motivazione precedentemente esposta), si ottiene un TPR o FPR pari a 0/0, ossia a un valore indeterminato, indicato nelle successive tabelle con ND. Statisticamente, questi casi sono rari e riguardano per lo più il FPR, poiché generalmente i numeri di falsi positivi e di veri negativi sono numericamente inferiori rispetto ai veri positivi e falsi negativi. Essendo in ogni caso ciò influenzato da un bias di identificazione di forza maggiore, si è deciso di non penalizzare alcuni casi specifici: in particolare, quando il TPR assume un valore significativo (e diverso dai casi estremi di 0 o 1) e il FPR assume un risultato indeterminato, si è deciso di considerare quest'ultimo come se fosse pari a 0, mentre in tutti gli altri casi il risultato resta di tipo ND.

Considerando le limitazioni sopra esposte, si è dunque deciso di riportare nelle tabelle che riguardano i TPR e i FPR tutti i risultati ottenuti, ma di creare ed esporre le rispettive OWASP Scorecard solo qualora almeno due strumenti su sei presentassero la seguente caratteristica: avere almeno uno dei due tassi diverso da 0, da 1 e da ND.

5.2 Valutazioni totali e medie

In questo Paragrafo vengono riportati i valori numerici ottenuti per la vista totale, che ricordiamo essere quella in cui tutte le sette applicazioni a priorità alta vengono raggruppate come se fossero un'unica applicazione. In particolare, in Tabella 5.1, vengono indicati i valori rispettivamente del TPR e del FPR per ognuno dei sei strumenti. In Figura 5.1 viene inoltre riportata la corrispondente OWASP Scorecard, con le indicazioni del parametro globale di valutazione tra parentesi in legenda.

	TPR	**FPR**
Strumento 1	0.89	0.35
Strumento 2	0.87	0.68
Strumento 3	0.88	0.56
Strumento 4	0.59	0.20
Strumento 5	0.48	0.40
Strumento 6	0.80	0.50

Tabella 5.1: Valori di TPR e FPR per l'insieme di tutte le vulnerabilità presenti nella code base

Vengono qui inoltre riportati i risultati separati in base alla pericolosità delle segnalazioni riportate dagli strumenti: la Tabella 5.2 contiene i valori dei TPR e FPR rispettivamente per le vulnerabilità di pericolosità alta e bassa (indicati con H e L), mentre le Tabelle 5.2 e 5.3 ne contengono le corrispondenti Scorecards con i valori di distanza dalla diagonale.

	TPR_H	**FPR_H**	**TPR_L**	**FPR_L**
Strumento 1	0.94	0.48	0.56	0.00
Strumento 2	0.87	0.67	ND	1.00
Strumento 3	0.78	0.58	1.00	0.00
Strumento 4	0.61	0.00	0.40	1.00
Strumento 5	0.42	0.40	1.00	ND
Strumento 6	0.74	0.71	1.00	0.00

Tabella 5.2: Valori dei TPR e FPR per l'insieme di tutte le vulnerabilità di pericolosità alta e bassa contenute nella codebase

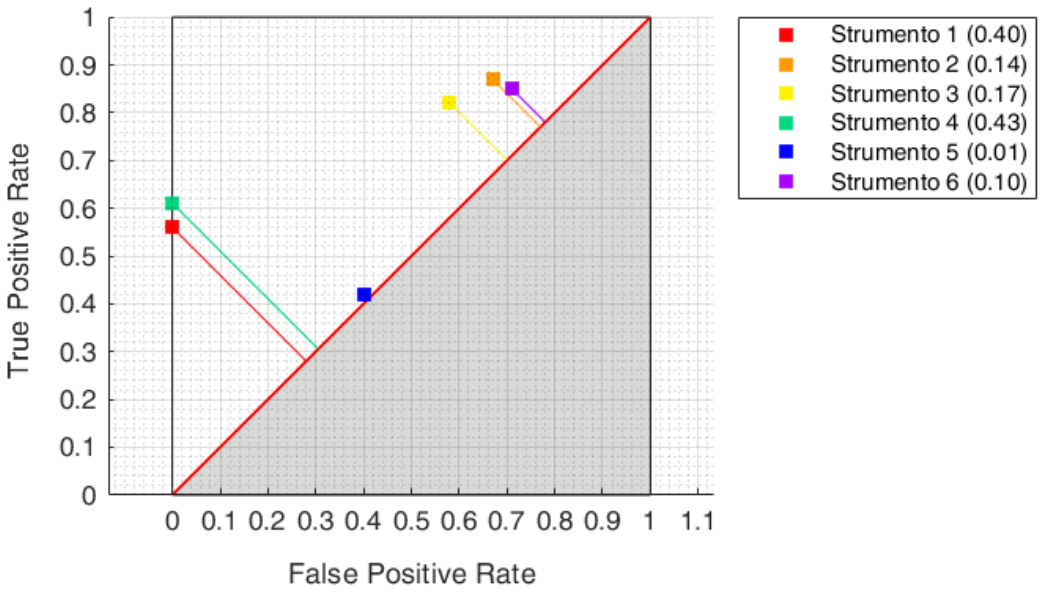

Figura 5.1: Scorecard relativa all'intera codebase

Si può notare come le Figure 5.1 e 5.2, che riportano rispettivamente tutte le vulnerabilità e solo quelle di pericolosità alta, siano pressappoco simili, con gli strumenti 1 e 4 che, con score di distanza pari all'incirca a 0.4, staccano nettamente gli altri strumenti che si stabilizzano attorno a valori molto più bassi (pari quasi a 0 per lo strumento 5).

Va invece interpretato in maniera differente il grafico in Figura 5.3, che comprende tutte le vulnerabilità di pericolosità bassa: questo caso specifico risente infatti di entrambe le limitazioni espresse nella Sezione 5.1.
In particolare, la presenza di un campione troppo poco vasto determina dei casi di TPR e/o FPR estremizzati verso lo 0 o l'1, come nel caso dello strumento 6 o dello strumento 2; la presenza del del bias di individuazione delle vulnerabilità negative determina invece dei casi di totali di TN e/o FN nulli i quali, combinandosi al fattore precedente, producono dei risultati di TPR/FPR non determinati, e ciò risulta nella presenza di soli quattro strumenti su sei.
Questo caso specifico risente inoltre ulteriormente del fatto che nei conteggi non

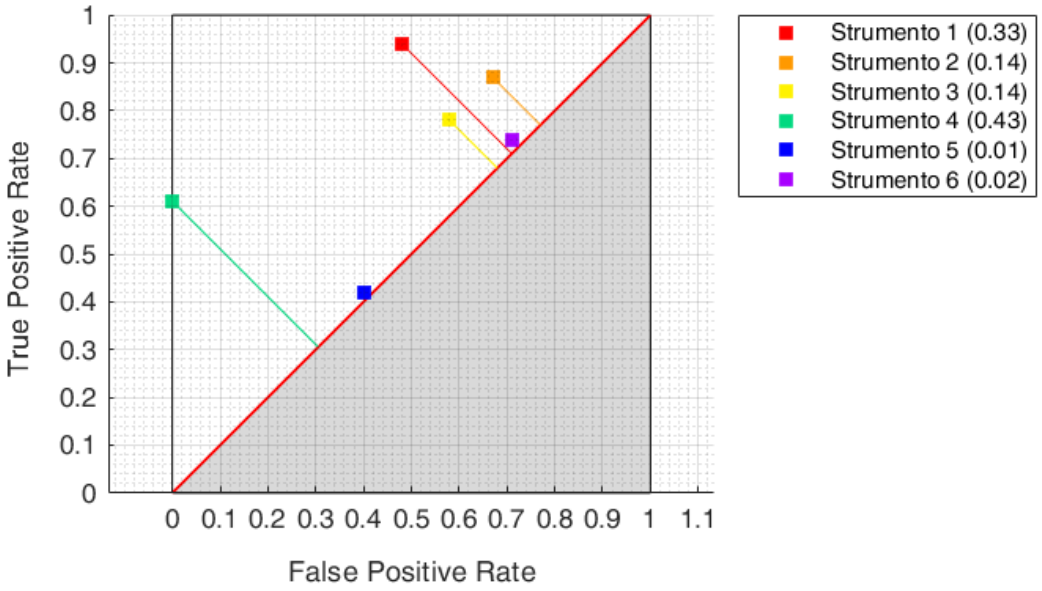

Figura 5.2: Scorecard relativa a tutte le vulnerabilità di pericolosità alta presenti nella codebase

siano presenti tutte le vulnerabilità di pericolosità bassa effettivamente esistenti nel programma, ma solo quelle che sono state giudicate di pericolosità medio-alta da almeno uno strumento (e bassa da altri), in base alle motivazioni espresse nella Sezione 3.2. Ciò risulta in un campione sensibilmente meno significativo rispetto agli altri due casi e, conseguentemente, in risultati molto differenti e in alcuni casi antitetici rispetto ai grafici precedenti, come si può notare dal valore negativo dello strumento 4.

In questa sede di presentazione dei risultati è stata compiuta inoltre un'ulteriore operazione rispetto alla vista totale presentata nel Paragrafo 4.1.1: si sono calcolati i valori medi dei vari TPR e FPR ottenuti dai sei strumenti sulle singole applicazioni.
Nelle Tabelle 5.3 e 5.4 vengono riportati i risultati di queste valutazioni mediate, comprendenti rispettivamente tutte le tipologie di vulnerabilità, e la suddivisione tra quelle di pericolosità alta e quelle di pericolosità bassa; nelle Figure 5.4, 5.5 e

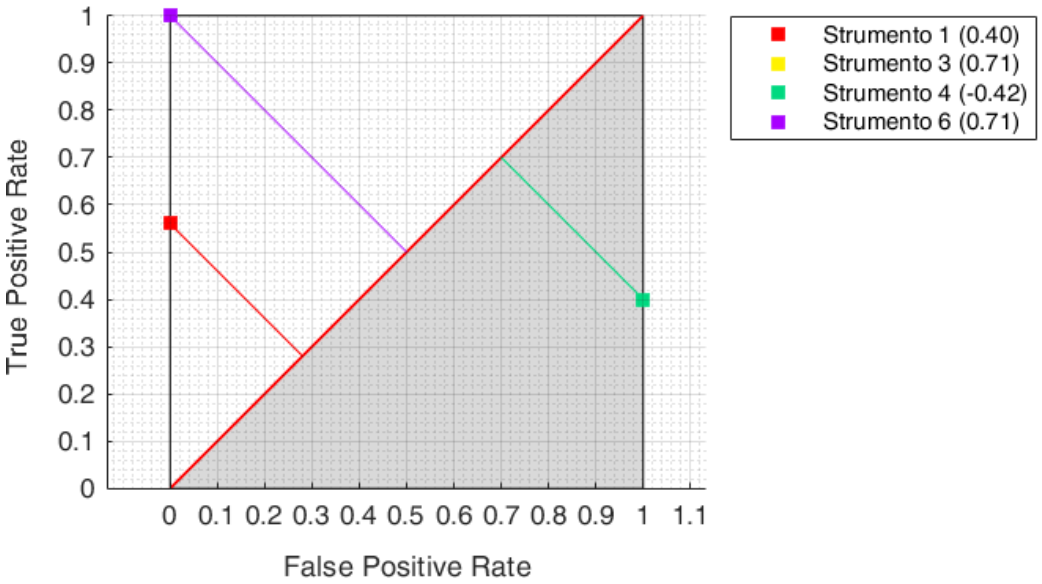

Figura 5.3: Scorecard relativa a tutte le vulnerabilità di pericolosità bassa presenti nella codebase

5.6 ne vengono illustrate invece le rispettive Scorecards.
Si può notare effettivamente una distribuzione più omogenea dei valori dei rate degli strumenti nelle Figure 5.4 e 5.5 rispetto alle corrispondenti Figure 5.1 e 5.2 della vista totale; per quanto riguarda la Figura 5.6, relativa alla media dei rate sulle vulnerabilità di pericolosità bassa, si può condurre un discorso analogo a quello sopra proposto nel caso della corrispondente Figura 5.3.

	TPR	**FPR**
Strumento 1	0.84	0.20
Strumento 2	0.86	0.63
Strumento 3	0.93	0.78
Strumento 4	0.69	0.14
Strumento 5	0.40	0.28
Strumento 6	0.61	0.21

Tabella 5.3: Valori medi di TPR e FPR nella codebase considerata

	TPR_H	**FPR_H**	**TPR_L**	**FPR_L**
Strumento 1	0.91	0.28	0.60	0.00
Strumento 2	0.86	0.56	ND	0.14
Strumento 3	0.91	0.80	1.00	0.00
Strumento 4	0.56	0.00	0.28	0.28
Strumento 5	0.40	0.28	0.28	ND -> 0
Strumento 6	0.61	0.26	0.28	0.00

Tabella 5.4: Valori medi di TPR e FPR nella codebase considerata, suddivisi per vulnerabilità di pericolosità alta e bassa

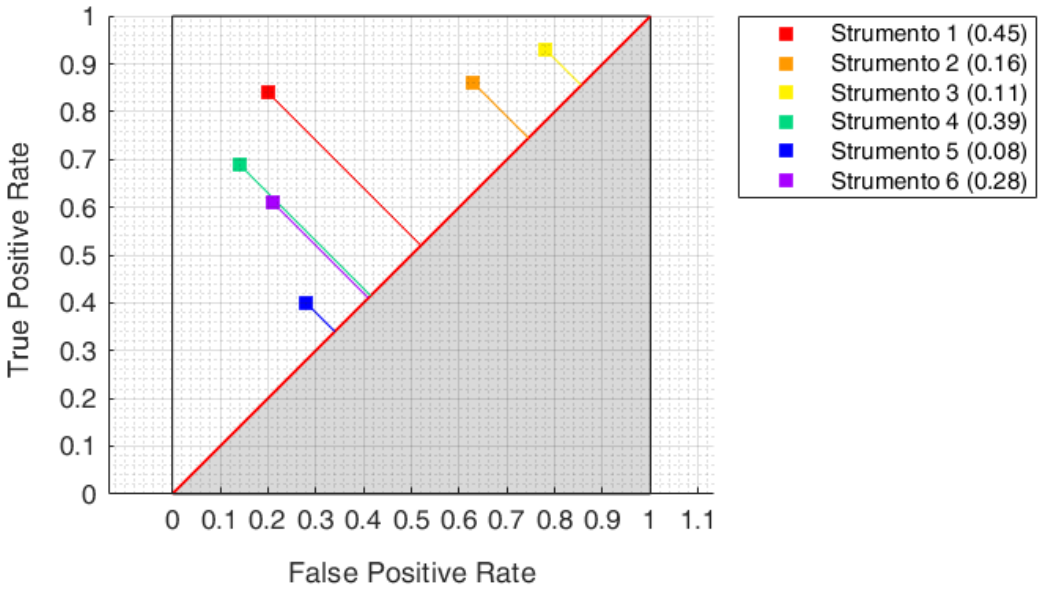

Figura 5.4: Scorecard relativa ai valori medi di tutte le vulnerabilità presenti nella codebase

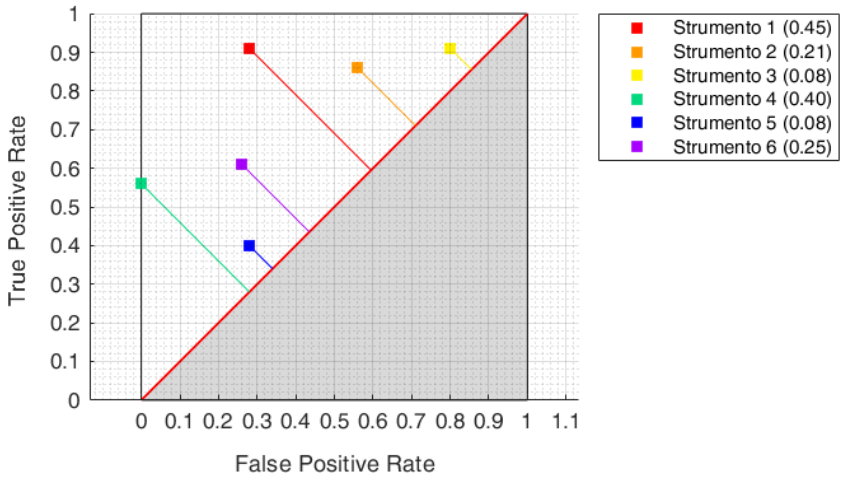

Figura 5.5: Scorecard relativa ai valori medi di tutte le vulnerabilità di pericolosità alta presenti nella codebase

Figura 5.6: Scorecard relativa ai valori medi di tutte le vulnerabilità di pericolosità bassa presenti nella codebase

5.3 Valutazioni per singola App

In questa Sezione si riprenderà la vista per singola applicazione, analizzando i risultati raggiunti per le sette applicazioni a priorità alta.
Nella Tabella 5.5 e nella Figura 5.7 sono riportati i risultati dei rate e la Scorecard ottenuti per "cwa-server", l'applicazione con più LOC della codebase [11]: si può notare la mancanza dello strumento 5 nella rappresentazione grafica, e il fatto che quasi tutti gli altri strumenti si attestino su valori di distanza medi.

	TPR	**FPR**
Strumento 1	0.63	0.17
Strumento 2	1.00	0.50
Strumento 3	0.86	0.75
Strumento 4	0.50	0.00
Strumento 5	ND	ND
Strumento 6	0.67	1.00

Tabella 5.5: Risultati dei rate relativi all'applicazione "cwa-server"

Nella Tabella 4.5 e nella Figura 5.8 sono invece riportati i valori di rate e la Scorecard per l'applicazione "dm-store" [24]: anche qui lo strumento 5 non è risultato idoneo alla rappresentazione, mentre si può notare una divisione netta tra alcuni strumenti che raggiungono valori di performance molto elevati e altri che invece presentano una distanza dalla diagonale molto ridotta: ciò è probabilmente dovuto alla tipologia di vulnerabilità maggiormente contenute nel programma, individuabili più da alcuni strumenti che da altri.

	TPR	**FPR**
Strumento 1	0.60	0.58
Strumento 2	0.40	0.43
Strumento 3	1.00	0.11
Strumento 4	1.00	0.00
Strumento 5	1.00	ND
Strumento 6	1.00	0.50

Tabella 5.6: Risultati dei rate relativi all'applicazione "dm-store"

Seguono nella Tabella 5.7 e nella Figura 5.9 i risultati per "employee management" [17]: in questo caso è lo strumento 6 a mancare, mentre per quanto riguarda gli strumenti 2 e 5 si può riscontrare quanto discusso nella Sezione 5.1 a proposito

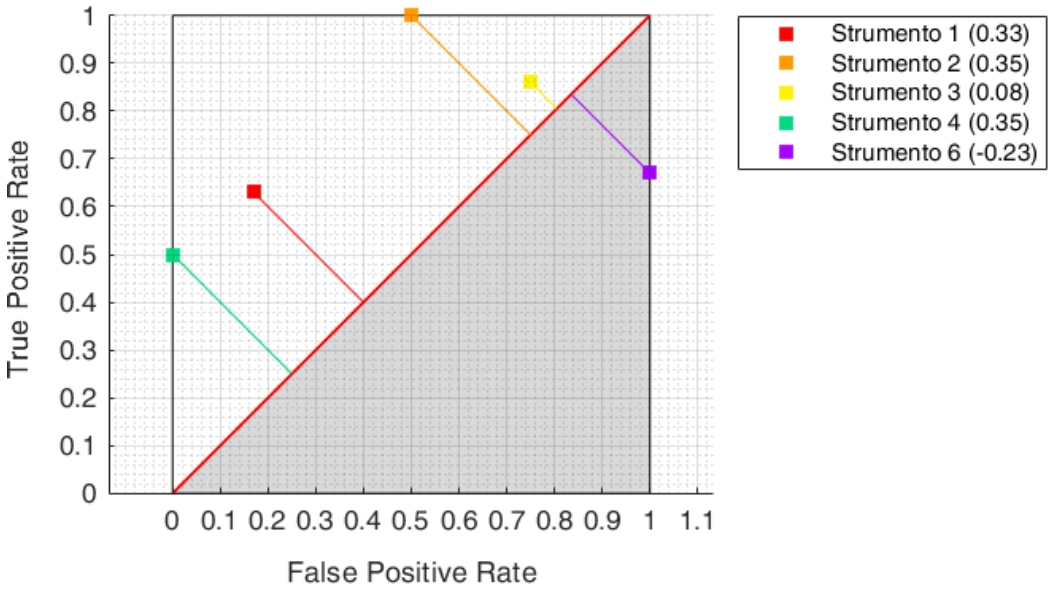

Figura 5.7: Scorecard relativa all'applicazione "cwa-server"

dei risultati fortemente polarizzati intorno allo 0 o all'1. Inoltre nel caso dello strumento 4, il FPR è passato da un valore di ND a 0 (per le motivazioni descritte nella Sezione 5.1).

	TPR	**FPR**
Strumento 1	1.00	0.13
Strumento 2	1.00	1.00
Strumento 3	1.00	0.63
Strumento 4	0.63	ND->0
Strumento 5	0.00	1.00
Strumento 6	ND	ND

Tabella 5.7: Risultati dei rate relativi all'applicazione "employee management"

In Tabella 5.8 e in Figura 5.10 sono riportati i valori dei rate e la Scorecard

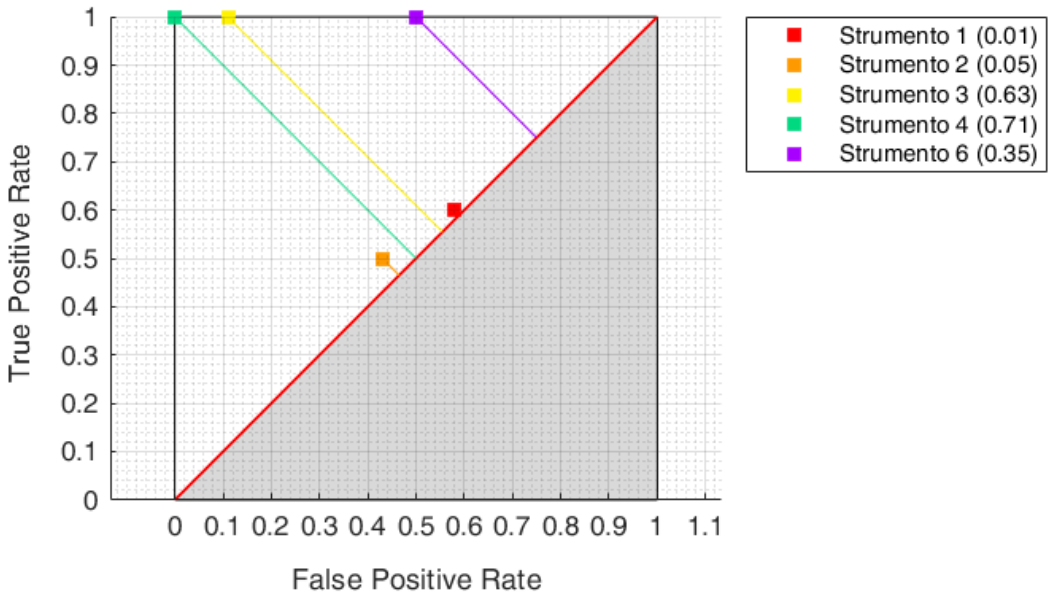

Figura 5.8: Scorecard relativa all'applicazione "dm-store"

dell'applicazione "jcart" [23]: si tratta di una situazione molto simile a quella della Tabella 5.6 e Figura 5.8 (riguardanti l'applicazione "dm-store"), caratterizzata dall'assenza dello strumento 5 e dalla divisione netta tra strumenti estremamente performanti e strumenti con punteggi prossimi allo 0 (o addirittura, lievemente negativi). Anche in questo caso, allo strumento 4 in fase di elaborazione dei risultati è stato assegnato il FPR nullo, mentre originariamente presentava un valore non definito.

Si passa ora ai risultati riguardanti l'applicazione "Poplar" [20], contenuti nella Tabella 5.9 e Figura 5.11: probabilmente a causa del campione relativamente ridotto di vulnerabilità individuate, solo gli strumenti 1 e 2 sembrano riprodurre dei risultati statisticamente rilevanti.

Nella Tabella 5.10 e nella Figura 5.12 vengono illustrati i risultati ottenuti nel caso di "Vulnerable App", l'applicazione OWASP volutamente vulnerabile [26]: è necessario evidenziare che quest'applicazione rappresenta un caso estremamente particolare all'interno dell'analisi, in cui tutti i FPR dei vari strumenti sono non definiti. Questo, come discusso nella Sezione 5.1, è dovuto al bias di cui soffre

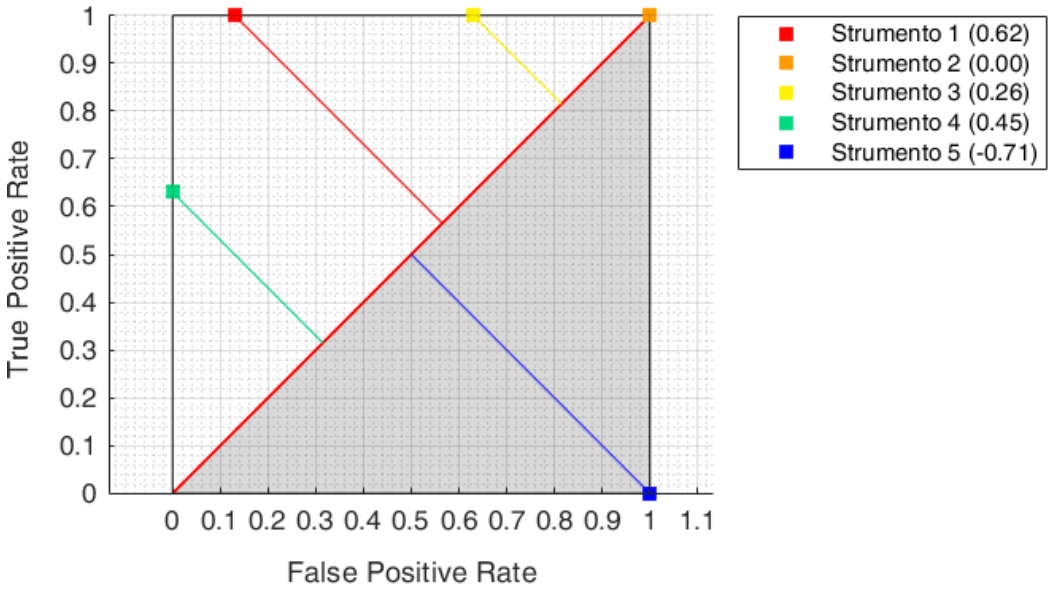

Figura 5.9: Scorecard relativa all'applicazione "employee management"

	TPR	**FPR**
Strumento 1	0.91	0.00
Strumento 2	0.90	0.00
Strumento 3	1.00	1.00
Strumento 4	0.18	ND->0
Strumento 5	ND	ND
Strumento 6	1.00	0.00

Tabella 5.8: Risultati dei rate relativi all'applicazione "jcart"

l'individuazione dei casi negative, ma non solo: essendo molte delle vulnerabilità introdotte deliberatamente nel codice, è logicamente più difficile che gli strumenti individuino dei falsi positivi; l'assenza dei falsi positivi, combinata all'assenza di veri negativi, risulta dunque nei valori di FPR pari a ND per tutti gli strumenti. Tuttavia, come specificato ancora nella Sezione 5.1, in questi casi in cui invece il

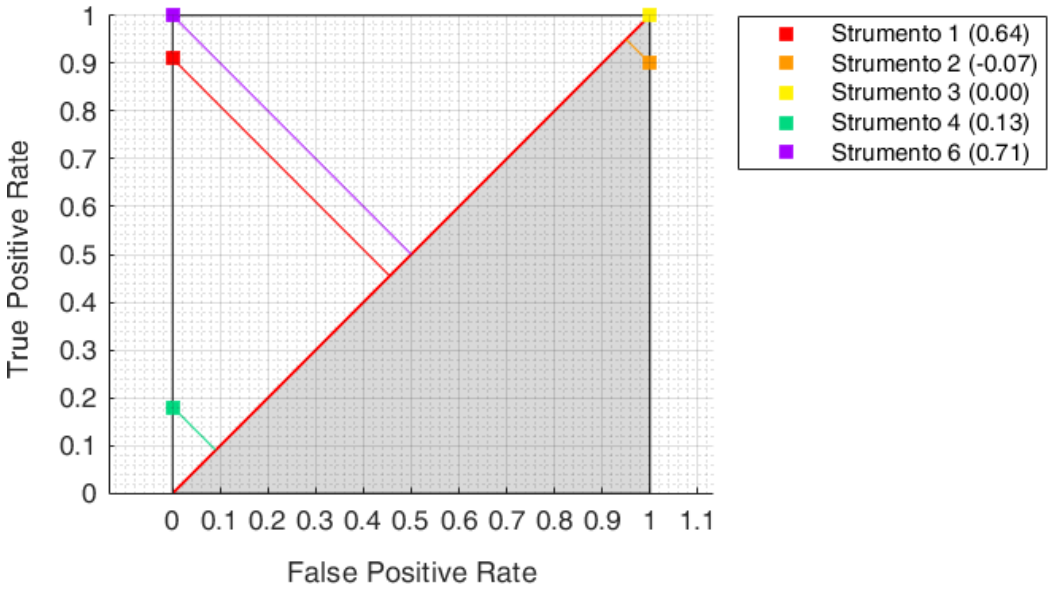

Figura 5.10: Scorecard relativa all'applicazione "jcart"

	TPR	**FPR**
Strumento 1	0.95	0.50
Strumento 2	1.00	0.50
Strumento 3	1.00	1.00
Strumento 4	1.00	1.00
Strumento 5	1.00	1.00
Strumento 6	ND	ND

Tabella 5.9: Risultati dei rate relativi all'applicazione "Poplar"

TPR assume dei valori statisticamente ragionevoli, si sono considerati i valori di FPR come pari a 0, in modo da poter comunque effettuare un confronto diretto tra gli strumenti basandosi sul TPR: in quest'ottica, il grafico in Figura 5.12 si può interpretare come una scala discendente di indici di prestazione, con gli strumenti 1 e 4 in testa, e gli strumenti 5 e 6 in coda.

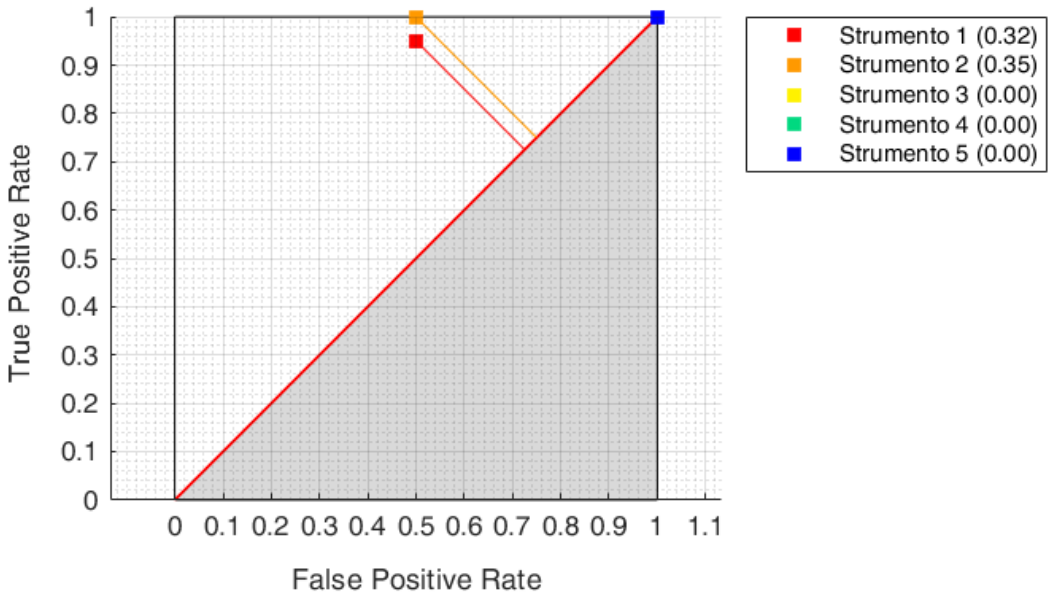

Figura 5.11: Scorecard relativa all'applicazione "Poplar"

	TPR	**FPR**
Strumento 1	0.88	ND -> 0
Strumento 2	0.61	ND -> 0
Strumento 3	0.60	ND -> 0
Strumento 4	0.72	ND -> 0
Strumento 5	0.11	ND -> 0
Strumento 6	0.38	ND -> 0

Tabella 5.10: Risultati dei rate relativi all'applicazione "Vulnerable App"

Infine, vengono presentati in Tabella 5.11 e in Figura 5.13 i risultati relativi all'applicazione "yugastore" [15], statisticamente simili a quelli riguardanti l'applicazione "Poplar", con un paio di strumenti che appaiono molto polarizzati ed altri che si attestano su valori di distanza importanti, sia in positivo che in negativo. Anche in questo caso, allo strumento 4 è stato assegnato d'ufficio un FPR nullo,

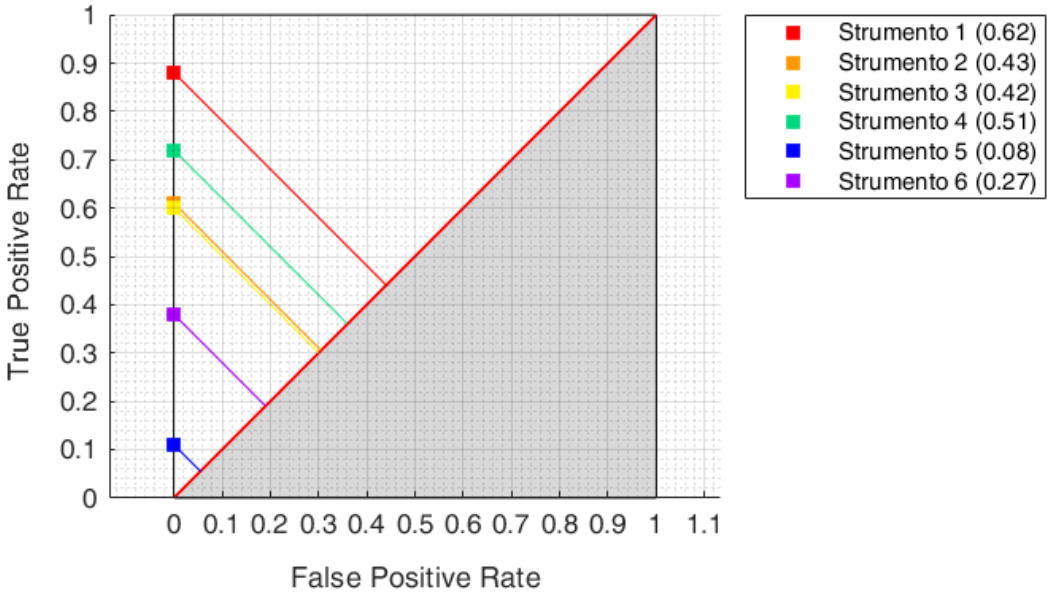

Figura 5.12: Scorecard relativa all'applicazione "Vulnerable App"

invece che un valore non definito.

	TPR	**FPR**
Strumento 1	0.92	0.00
Strumento 2	1.00	1.00
Strumento 3	1.00	1.00
Strumento 4	0.80	ND -> 0
Strumento 5	0.71	0.00
Strumento 6	1.00	ND

Tabella 5.11: Risultati dei rate relativi all'applicazione "yugastore"

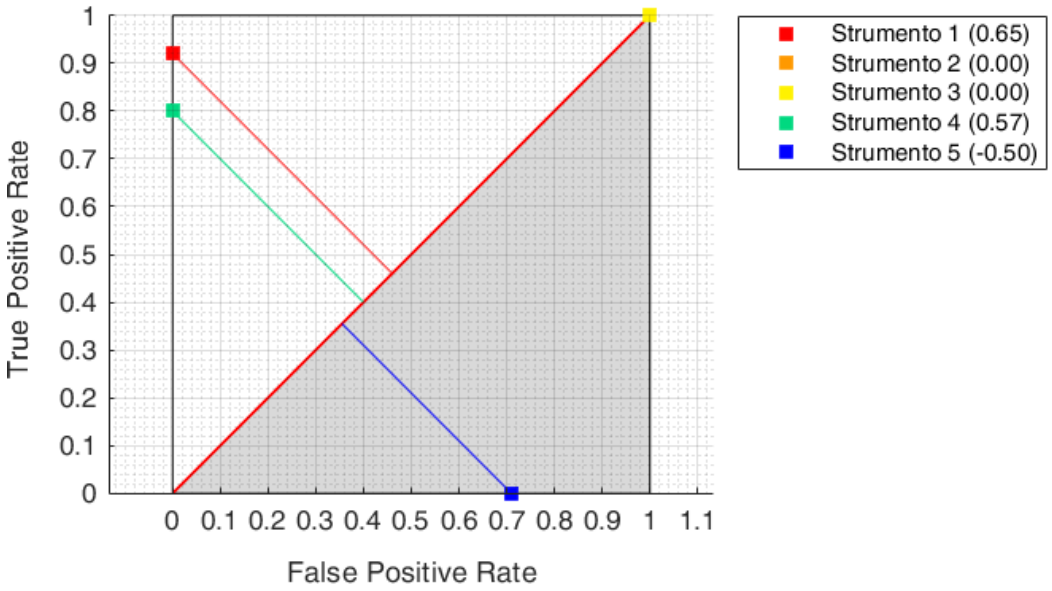

Figura 5.13: Scorecard relativa all'applicazione "yugastore"

5.4 Valutazioni per categorie di vulnerabilità

In questa Sezione vengono illustrati i risultati ottenuti riprendendo la vista per categorie di vulnerabilità, già presentata nel Paragrafo 4.1.3: in particolare, nelle Tabelle dalla 5.12 alla 5.17 si riportano i valori ottenuti per tutte le otto categorie semplici, già precedentemente indicate come "Non-OWASP". A causa delle forte specificità delle categorie di vulnerabilità, che porta al filtraggio di un campione troppo ridotto, si può notare come quasi nessuna di esse presenti i requisiti di cui alla Sezione 5.1 descritti come necessari alla rappresentazione grafica.
Si riporta in Figura 5.14 la Scorecard dell'unica categoria facente eccezione, quella relativa ad "Other Vulnerabilities": essendo infatti questa una tipologia contenitrice di tutte le categorie di vulnerabilità minori, è rappresentativa di un campione più ampio ed ha dato luogo a risultati statisticamente più significativi. Come emerge dal grafico, solo lo strumento 1 è stato in ogni caso in grado di raggiungere valori alti di prestazioni, mentre gli strumenti 2 e 5 si attestano a prestazioni minime e lo strumento 3 scende in negativo.

Viene inoltre illustrato in questa sede il grafico della Scorecard (Figura 5.15) corrispondente all'insieme di tutte le vulnerabilità comprese nelle precedenti otto categorie semplici (e dunque a un campione evidentemente più ampio), in cui gli strumenti 6, 1 e 3 raggiungono prestazioni medio alte e nessuno strumento presenta prestazioni negative.

	TPR	**FPR**
Strumento 1	1.00	ND
Strumento 2	ND	ND
Strumento 3	0.00	ND
Strumento 4	1.00	ND
Strumento 5	ND	ND
Strumento 6	0.00	ND

Tabella 5.12: Rate per "Command Injection"

	TPR	**FPR**
Strumento 1	0.00	ND
Strumento 2	ND	ND
Strumento 3	1,00	ND
Strumento 4	ND	ND
Strumento 5	ND	ND
Strumento 6	ND	ND

Tabella 5.13: Rate per "LDAP Injection"

	TPR	FPR
Strumento 1	0.66	1.00
Strumento 2	ND	ND
Strumento 3	0.00	0.00
Strumento 4	1.00	0.00
Strumento 5	ND	ND
Strumento 6	0.00	0.00

Tabella 5.14: Rate per "Path Traversal"

	TPR	FPR
Strumento 1	1.00	0.46
Strumento 2	ND	ND
Strumento 3	0.37	0.54
Strumento 4	0.75	0.00
Strumento 5	ND	ND
Strumento 6	ND	ND

Tabella 5.15: Rate per "SQL Injection"

	TPR	FPR
Strumento 1	1.00	1.00
Strumento 2	ND	ND
Strumento 3	1.00	1.00
Strumento 4	1.00	1.00
Strumento 5	ND	ND
Strumento 6	ND	ND

Tabella 5.16: Rate per "Weak Hash"

	TPR	FPR
Strumento 1	1.00	0.00
Strumento 2	ND	1.00
Strumento 3	1.00	0.00
Strumento 4	ND	ND
Strumento 5	ND	ND
Strumento 6	ND	ND

Tabella 5.17: Rate per "Weak Random"

	TPR	FPR
Strumento 1	1.00	1.00
Strumento 2	1.00	1.00
Strumento 3	1.00	0.00
Strumento 4	0.48	ND
Strumento 5	ND	ND
Strumento 6	1.00	0.00

Tabella 5.18: Rate per "Spring related"

	TPR	FPR
Strumento 1	0.78	0.17
Strumento 2	0.88	0.86
Strumento 3	0.50	1.00
Strumento 4	ND	ND
Strumento 5	0.45	0.40
Strumento 6	1.00	ND

Tabella 5.19: Rate per "Other Vulnerabilities"

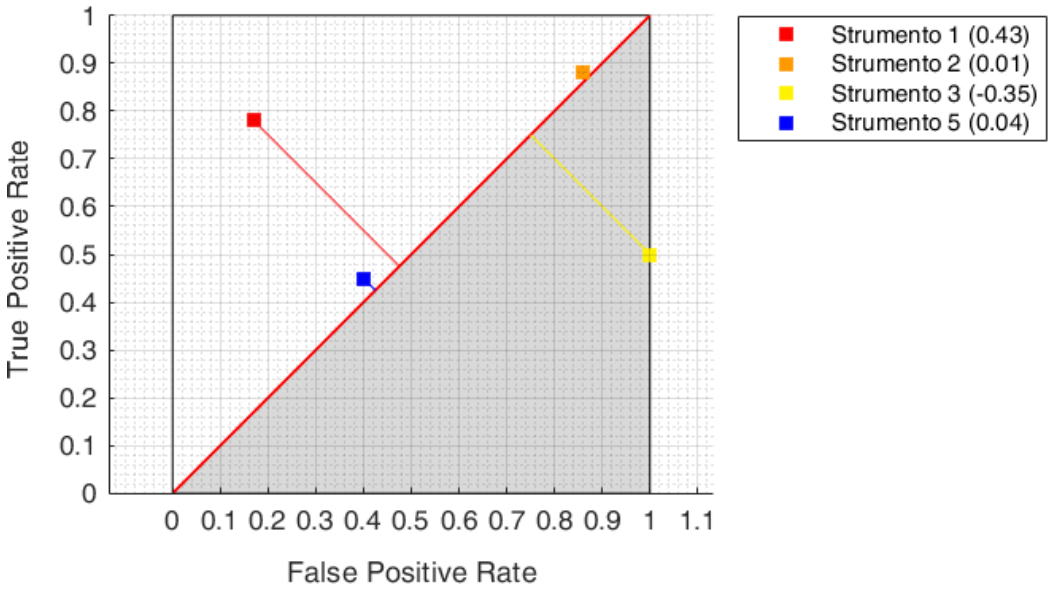

Figura 5.14: Scorecard relativa alla categoria "Other Vulnerabilities"

Si passa ora a illustrare, nelle Tabelle dalla 5.20 alla 5.27, i rate ottenuti per le otto categorie di tipo OWASP, sempre tenendo conto che non si sono riscontrate vulnerabilità delle tipologie "A05" e "A06". Anche in questo caso, considerata la specificità (minore rispetto alle categorie non OWASP, ma pur sempre elevata),vi sono pochi casi in cui il campione risulta sufficientemente ampio da poter dare luogo a una Scorecard.
Se ne riportano dunque in Figura 5.16 e 5.17 i due esempi più significativi, relativi rispettivamente alla categoria “A01” e “A03”. In particolare, per quanto riguarda la categoria “A01”, si riscontrano performance considerevoli sugli strumenti 3,4 e 6; nel caso della “A03” d’altro canto solo gli strumenti 1 e 4 raggiungono buone prestazioni, mentre lo strumento 3 scivola sotto la soglia del random guessing.

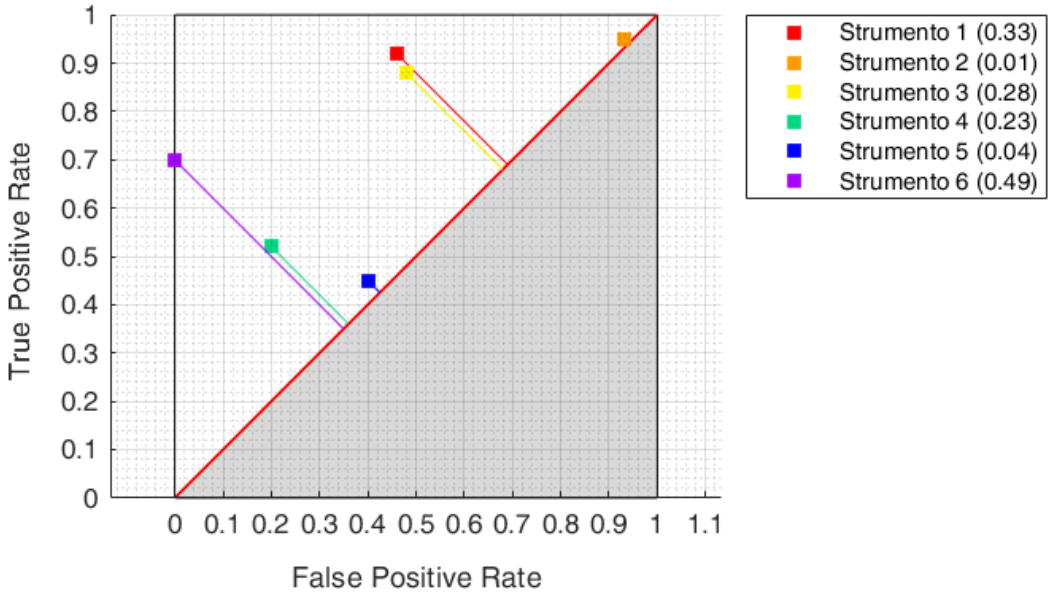

Figura 5.15: Scorecard relativa all'insieme delle categorie cosiddette "Non OWASP"

	TPR	FPR
Strumento 1	0.98	1.00
Strumento 2	ND	ND
Strumento 3	0.94	0.25
Strumento 4	1.00	0.00
Strumento 5	ND	ND
Strumento 6	0.75	0.00

Tabella 5.20: Rate per "A01- Broken Access Control"

	TPR	FPR
Strumento 1	1.00	0.33
Strumento 2	ND	1.00
Strumento 3	1.00	0.33
Strumento 4	1.00	1.00
Strumento 5	ND	ND
Strumento 6	ND	ND

Tabella 5.21: Rate per "A02- Cryptographic Failures"

	TPR	FPR
Strumento 1	0.96	0.46
Strumento 2	1.00	ND
Strumento 3	0.36	0.50
Strumento 4	0.80	0.00
Strumento 5	ND	ND
Strumento 6	ND	0.00

Tabella 5.22: Rate per "A03- Injections"

	TPR	FPR
Strumento 1	1.00	0.00
Strumento 2	ND	ND
Strumento 3	1.00	1.00
Strumento 4	1.00	ND
Strumento 5	ND	ND
Strumento 6	ND	0.00

Tabella 5.23: Rate per "A04- Insecure Design"

	TPR	FPR
Strumento 1	0.50	0.00
Strumento 2	ND	ND
Strumento 3	1.00	1.00
Strumento 4	1.00	ND
Strumento 5	ND	ND
Strumento 6	ND	ND

Tabella 5.24: Rate per "A07- Ident/AuthN failures"

	TPR	FPR
Strumento 1	ND	ND
Strumento 2	1.00	ND
Strumento 3	ND	ND
Strumento 4	0.23	ND->0
Strumento 5	ND	ND
Strumento 6	ND	ND

Tabella 5.25: Rate per "A08- Software/data failures"

	TPR	FPR
Strumento 1	1.00	ND
Strumento 2	ND	ND
Strumento 3	ND	ND
Strumento 4	ND	ND
Strumento 5	ND	ND
Strumento 6	ND	ND

Tabella 5.26: Rate per "A09- Monitoring failures"

	TPR	FPR
Strumento 1	1.00	ND
Strumento 2	ND	ND
Strumento 3	ND	ND
Strumento 4	1.00	ND
Strumento 5	ND	ND
Strumento 6	ND	ND

Tabella 5.27: Rate per "A10- SSRF"

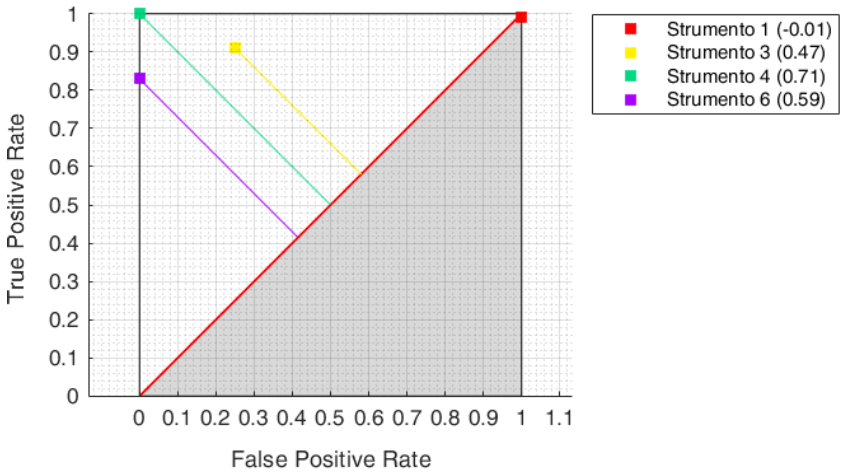

Figura 5.16: Scorecard relativa alla categoria "A01"

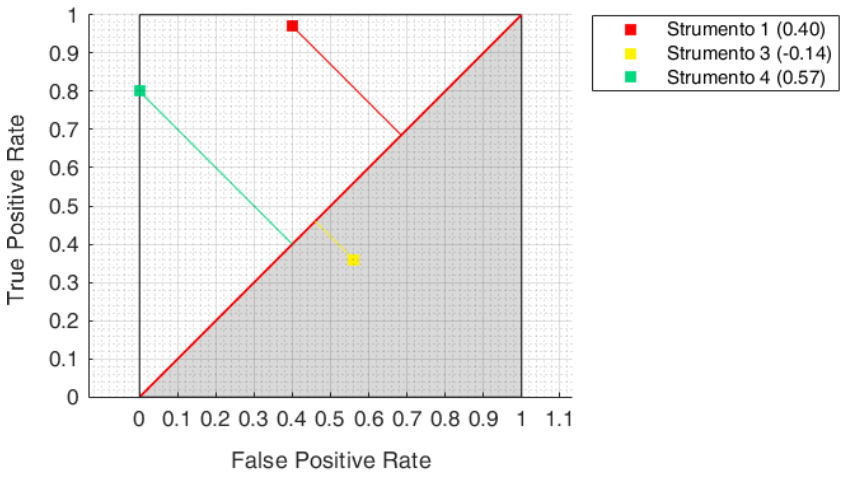

Figura 5.17: Scorecard relativa alla categoria "A03"

5.5 Valutazioni totali per categorie OWASP

In quest'ultima Sezione si illustrano i risultati relativi al totale delle vulnerabilità rientranti nelle otto categorie OWASP, riprendendo la vista del Paragrafo 4.1.4. In Tabella 5.28 vengono riportati i TPR e FPR per i vari strumenti, mentre la Figura 5.18 contiene la relativa Scorecard. Si può notare che, considerato il campione abbastanza ampio, nessuno strumento si trova sotto la soglia del random guessing; gli strumenti 1 e 6 raggiungono prestazioni considerevoli, seguiti dagli strumenti 3 e 4.

	TPR	**FPR**
Strumento 1	0.98	0.42
Strumento 2	1.00	1.00
Strumento 3	0.87	0.53
Strumento 4	0.72	0.20
Strumento 5	ND	ND
Strumento 6	0.77	0.00

Tabella 5.28: Valori di TPR e FPR per l'insieme di tutte le vulnerabilità contenute nelle categorie OWASP

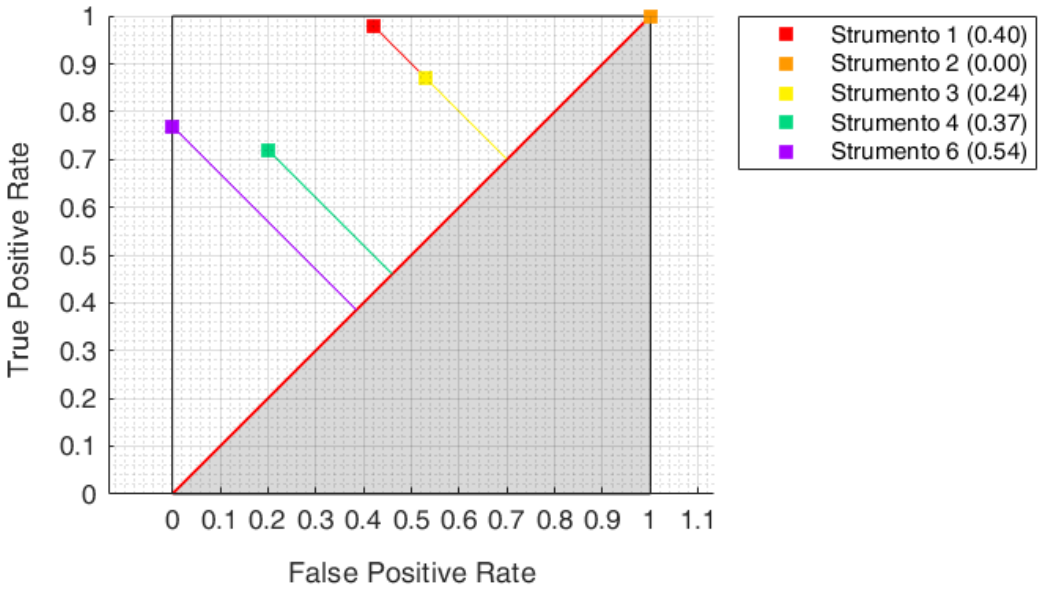

Figura 5.18: Scorecard relativa all'insieme delle vulnerabilità rientranti nelle categorie OWASP

5.6 Classifica generale

Si riporta infine in Tabella 5.29 la classifica totale delle performance degli strumenti, ottenuta mediante la media aritmetica dei vari valori di distanza contenuti nelle Figure di questo Capitolo.
Si ricorda che la distanza d si estende nel dominio compreso tra $\frac{-1}{\sqrt{2}}$ e $\frac{1}{\sqrt{2}}$, ossia tra circa -0.71 e 0.71, ma che uno strumento di individuazione delle vulnerabilità, per dichiararsi tale, dovrebbe presentare un punteggio positivo.
Si può dunque constatare come gli strumenti 1, 4 e 6 presentino valori medi di punteggi medio-alti, attestandosi nell'intorno della metà del range ammissibile; gli strumenti 3,2 e 5 presentano invece valori medi decisamente inferiori, seppur comunque mediamente positivi.
Come verrà discusso nel successivo Capitolo conclusivo, considerando la natura non estremamente professionale degli strumenti adottati, e tenendo presente come il campione analizzato provenga da applicazioni comuni e non da appositi Benchmark valutativi, i risultati raggiunti possono essere considerati complessivamente in linea con le aspettative e, in taluni casi, leggermente superiori.

Posizione	**Strumento**	**Punteggio Medio**
1	Strumento 1	0.40
2	Strumento 4	0.34
3	Strumento 6	0.33
4	Strumento 3	0.20
5	Strumento 2	0.13
6	Strumento 5	0.05

Tabella 5.29: Classifica generale delle performance degli strumenti

Capitolo 6

Conclusioni e sviluppi futuri

Dalle analisi condotte in questo lavoro di tesi sono emersi risultati complessivamente in linea con le aspettative, sia per quanto riguarda la ricerca di vulnerabilità che le performance degli strumenti.

Valutazioni complessive

Considerando la natura Open Source e destinazione d'uso diversa da quella di Benchmark delle applicazioni adottate, si ritene che la quantità e la pericolosità delle vulnerabilità individuate siano conformi alla norma delle applicazioni web e, in taluni casi, inferiori alla media: se da un lato ciò ha condizionato le scelte effettuate (poiché ai fini delle analisi le applicazioni più sicure sono state chiaramente considerate meno interessanti), dall'altro ciò rappresenta un segno di come la programmazione moderna si stia sempre più allineando ai canoni della sicurezza informatica anche nei casi di applicazioni comuni, non considerabili particolarmente a rischio.

Ancora più rilevante è la considerazione per cui, tenendo presente la natura non altamente professionale degli strumenti utilizzati, si ritengono i risultati della Tabella 5.29 perfettamente in linea, e in alcuni casi leggermente superiori, alle attese; gli strumenti sono stati infatti complessivamente in grado di fornire statistiche adeguate al contesto di analisi, non scendendo sotto la soglia di random guessing a meno di situazioni singolari (in cui in ogni caso gli output sono stati fortemente influenzati dalla qualità del campione, più che dalla bontà degli strumenti), e anzi raggiungendo talvolta prestazioni considerevoli.

Questo lavoro di tesi non è tuttavia che il punto di partenza per ulteriori studi riguardanti l'applicazione della SAST Analysis ai moderni contesti di programmazione distribuita.

Classificazione dei risultati e possibili miglioramenti

Scendendo più nel dettaglio, si vuole qui condurre un'analisi critica dei risultati ottenuti, illustrandone, ove necessario, possibili spunti di miglioramento.

Risultati totali e medi

Per risultati di insieme si intendono quelli relativi alle analisi condotte tenendo come code base tutte le segnalazioni provenienti dagli strumenti: rientrano qui i risultati totali e medi calcolati su tutte le vulnerabilità, solo su quelle di pericolosità alta e solo su quelli di pericolosità bassa. Questa casistica è dunque quella comprendente il campione più vasto, e ciò porta ad attutire gli effetti dell'avere un campione troppo poco significativo (descritti nella sezione 5.1) come ad esempio l'estremizzazione dei valori di TPR e/o FPR.
Si può notare come i grafici rientranti nelle valutazioni totali e medie per tutte le vulnerabilità e per le vulnerabilità a pericolosità alta, siano in assoluto i migliori di questo lavoro di tesi, segno che un campione statisticamente significativo porta inevitabilmente a risultati più accurati.

Un discorso differente va invece condotto per quanto riguarda le vulnerabilità di pericolosità bassa: queste ultime sono infatti molto inferiori in numero, poiché si è deciso di impostare la raccolta dei dati basandosi su tutte quelle vulnerabilità che fossero state giudicate di pericolosità alta da almeno uno strumento (ed eventualmente bassa da altri). A un campione minore conseguono logicamente dei risultati fortemente meno accurati, in cui la distanza d risulta essere talvolta negativa, ed i valori di TPR ed FPR estremizzati verso lo 0 o l'1.

Qualora si fosse interessati ad esempio ad approfondire la presenza e la distribuzione di vulnerabilità minori, considerate di pericolosità inferiore, all'interno di applicazioni comuni, si potrebbe dunque migliorare questa parte dell'analisi adottando un protocollo di registrazione dei risultati che tenga conto di *tutte* le segnalazioni restituite dagli strumenti. A questo scopo, si potrebbero anche valutare in modo più approfondito le nove applicazioni giudicate a priorità media: queste ultime infatti, pur non riportando particolari segnalazioni di vulnerabilità pericolose, presentano comunque in output molte segnalazioni di carattere minore.

Risultati per applicazione

Con "risultati per applicazione" si intendono i risultato ottenuti considerando come code base le vulnerabilità relative alle sette singole applicazioni a priorità alta.
Questa casistica viene considerata da chi scrive come quella che ha portato a risultati meno significativi dal punto di vista statistico: come si è già accennato, infatti, le

applicazioni scelte provengono da ambiti comuni (con l'eccezione di VulnerableApp) e non hanno alcuna finalità di Benchmark: ne consegue che, sperabilmente, esse siano poco soggette a vulnerabilità particolarmente pericolose, che sono quelle sui cui si è concentrata maggiormente questa ricerca.

Si potrebbe dunque considerare di variare la codebase in oggetto, prendendo ad esempio in considerazione applicazioni che non adottino necessariamente il framework SpringBoot (al fine di poter avere maggiori possibilità di scelta), o che siano specificatamente pensate come Benchmark per la SAST Analysis: questa opzione consentirebbe infatti, a patto di variare leggermente il fine dell'analisi verso obiettivi diversi dalle applicazioni comuni, di avere accesso ad un campione più ampio e statisticamente più significativo di vulnerabilità.

Risultati per categorie di vulnerabilità

Con "risultati per categorie di vulnerabilità" si intendono quelli ottenuti considerando sia le otto categorie "semplici" che le otto categorie OWASP, prese sia singolarmente che complessivamente. Questa casistica ha prodotto risultati molto variabili, tra il molto soddisfacente e il poco significativo dal punto di vista statistico.

In particolare, vengono considerati da chi scrive come molto soddisfacenti i casi relativi ai due *insiemi* di vulnerabilità, ossia tutte quelle semplici e tutte quelle OWASP: anche in questo caso si può dunque osservare il raggiungimento di buoni risultati qualora si parta da un campione più ampio.
Ci sono poi stati dei casi di risultati mediamente significativi, come quello relativo a tutte le vulnerabilità OWASP di tipo A01, e casi molto meno soddisfacenti, come quello relativo all'OWASP A03 o alla categoria "other vulnerabilities", i quali partono da una codebase molto più ristretta (seppur comprendendo, nei casi OWASP, anche le vulnerabilità proveniente dale nove applicazioni di riserva a priorità bassa).

Come precedentemente affermato, anche in questo caso ampliare la codebase oggetto di analisi può essere un ottimo spunto di miglioramento, ed in particolare potrebbe essere utile, per affinare il calcolo delle categorie OWASP, considerare applicazioni appositamente vulnerabili, come dei Benchmark. Si potrebbe anche porre attenzione a considerare applicazioni che comprendano le categorie "A05:2021 – Security Misconfiguration" e "A06:2021 – Vulnerable and Outdated Components", delle quali non è stata individuata nessuna occorrenza in questa sede.

Spunti di ampiamento della ricerca

Oltre ai possibili spunti di miglioramento sopra discussi, si elencano di seguito alcune possibili modalità di ampliamento delle analisi, adottabili da ricerche future complementari,che possano integrare le informazioni già ottenute con risultati di natura leggermente differente.

- Si potrebbe considerare di variare i requisiti di scelta degli strumenti, considerando opzioni commerciali specificatamente studiate al fine di ottenere prestazioni elevati nell'ambito dell'applicazione della SAST Analysis.
- Si potrebbero considerare gli stessi strumenti adottati in questa sede, ma in altre versioni di integrazione (come l'inserimento nelle pipelines di CI/CD) o adottandone altre licenze, non relative a versioni di prova o Educational.
- Si potrebbero studiare il comportamento degli strumenti utilizzati nell'ambito di altri linguaggi di programmazione: se da un lato Java continua ad essere uno dei linguaggi più diffusi specialmente lato backend, si potrebbe pensare di ampliare la ricerca a linguaggi di stampo più recente, anche al fine di poter anticipare l'individuazione di possibili aree di miglioramento di programmazione nell'ambito della sicurezza informatica.
- Si potrebbe considerare l'introduzione deliberata di vulnerabilità di varia entità e pericolosità all'interno del codice Open Source, per verificare se e quanto gli output degli strumenti possano risentire delle variazioni inserite.
- Un altro spunto interessante potrebbe consistere nel condurre una ricerca utilizzando gli stessi strumenti e le stesse applicazioni adottate in questa sede, ma sostituendo il parametro d di valutazione delle prestazioni con ad esempio i citati indice di correlazione di Metthews o F-measure: ciò potrebbe fornire informazioni rilevanti su quali siano le scale di misurazione da adottare preferibilmente a seconda delle caratteristiche degli strumenti.

In conclusione, vi è l'auspicio che questo lavoro di tesi abbia apportato un contributo in qualche maniera significativo al campo della valutazione delle prestazioni degli strumenti per l'Analisi Statica della sicurezza; ci si augura al tempo stesso che esso possa divenire in futuro un punto di partenza per ricerche più dettagliate, che mirino ad approfondire i risultati fin qui raggiunti e ad integrarli con valutazioni di natura affine o complementare.

Bibliografia

[1] *Forrester Report 2021*. 2021. URL: https://start.paloaltonetworks.com/2021-state-of-secops-forrester (cit. a p. 1).

[2] *The Economic Impacts of Inadeguate Infastructure for Software Testing*. 2022. URL: https://www.nist.gov/system/files/documents/director/planning/report02-3.pdf (cit. a p. 2).

[3] *SAST testing: how it works and why do you need it | Snyk*. URL: https://snyk.io/learn/application-security/static-application-security-testing/ (cit. a p. 2).

[4] *OWASP Benchmark*. URL: https://owasp.org/www-project-benchmark/ (cit. alle pp. 4, 35, 45, 47).

[5] *Juliet Test Suite*. URL: https://github.com/find-sec-bugs/juliet-test-suite (cit. a p. 4).

[6] *OWASP Source code Analysis Tools*. URL: https://owasp.org/www-community/Source_Code_Analysis_Tools (cit. a p. 5).

[7] *NIST Source Code Security Analyzers*. URL: https://www.nist.gov/itl/ssd/software-quality-group/source-code-security-analyzers (cit. a p. 5).

[8] *Common Weakness Enumeration*. URL: https://cwe.mitre.org/ (cit. a p. 7).

[9] *SANS Top 25 Software Errors*. URL: https://www.sans.org/top25-software-errors/ (cit. a p. 7).

[10] *OWASP Top 10:2021*. URL: https://owasp.org/Top10/it/ (cit. alle pp. 7, 18, 24).

[11] *cwa-server*. URL: https://github.com/corona-warn-app/cwa-server (cit. alle pp. 13, 61).

[12] *MyCollab*. URL: https://github.com/MyCollab/mycollab (cit. a p. 13).

[13] *web-application-for-donors*. URL: https://github.com/Habib-Aribane/web-application-for-donors (cit. a p. 13).

[14] *BookStoreApp-Distributed Application*. URL: https://github.com/devdcores/BookStoreApp-Distributed-Application (cit. a p. 13).

[15] *yugastore java*. URL: https://github.com/YugabyteDB-Samples/yugastore-java (cit. alle pp. 13, 66).

[16] *atsea-sample-shop-app*. URL: https://github.com/dockersamples/atsea-sample-shop-app (cit. a p. 13).

[17] *employee-management*. URL: https://github.com/mahi-mullapudi/employee_management (cit. alle pp. 13, 61).

[18] *my-moments*. URL: https://github.com/amrkhaledccd/my-moments (cit. a p. 13).

[19] *AppointmentScheduler*. URL: https://github.com/slabiak/AppointmentScheduler (cit. a p. 13).

[20] *Poplar*. URL: https://github.com/lvwangbeta/Poplar (cit. alle pp. 13, 63).

[21] *TravelWebApplication-Virtugo*. URL: https://github.com/pajaydev/TravelWebApplication-Virtugo (cit. a p. 13).

[22] *book-project*. URL: https://github.com/Project-Books/book-project (cit. a p. 13).

[23] *JCart*. URL: https://github.com/sivaprasadreddy/jcart (cit. alle pp. 13, 63).

[24] *document-management-store-app*. URL: https://github.com/hmcts/document-management-store-app (cit. alle pp. 13, 61).

[25] *spring-boot-react-blog*. URL: https://github.com/keumtae-kim/spring-boot-react-blog (cit. a p. 13).

[26] *VulnerableApp*. URL: https://github.com/SasanLabs/VulnerableApp (cit. alle pp. 13, 29, 63).