UNIVERSITÀ DEGLI STUDI DI SALERNO

Dipartimento di Informatica

Corso di Laurea Triennale in Informatica

TESI DI LAUREA

Predizione di vulnerabilità software tramite machine learning

RELATORE

Prof. **Fabio Palomba**

Università degli studi di Salerno

CANDIDATO

Gabriele Moscato

Matricola: 0512108099

Anno Accademico 2021-2022

"Ascolta quel che insegna la sapienza, cerca di capire la sua lezione. Ricerca la conoscenza e desidera la saggezza, come si desidera l'argento o si va in cerca di tesori" (La Bibbia).

Sommario

Questo studio si pone l'obiettivo di indagare sulle cause, sull'impatto economico e sociale causato dalle vulnerabilità software. Lo studio si focalizza principalmente su come e in che modo l'uso di modelli di machine learning posso essere utili per evitare e individuare vulnerabilità software.

Durante il mio lavoro di tesi ho analizzato alcuni studi svolti in passato in questo ambito e le prestazioni ottenute da quest'ultimi, sulla base di questi studi ho sviluppato un modello di predizione basato su machine learning capace di predirre la presenza di vulnerabilità software per applicazioni Java, il mio studio si conclude analizzando le prestazioni ottenute dal predittore.

Indice

Elenco delle figure

Elenco delle tabelle

CAPITOLO 1

Introduzione

1.1 Contesto applicativo

Con l'espansione e la crescita della digitalizzazione in tutti gli ambiti della nostra società, si è verificato un aumento significativo di fenomeni criminali in ambito digitale, la sicurezza informatica nasce con il proposito di contrastare e annullare questa tipologia di criminalità. La sicurezza informatica non ha come scopo solo quello di evitare attacchi informatici ma anche quello di offrire software o dispositivi capaci di essere robusti e affidabili, evitando che questi possano essere compromessi.
La crescita dello spazio cibernetico ovvero l'interconnessione delle infrastrutture informatiche [1] ha reso i sistemi informatici su cui si fonda più vulnerabili a possibili manipolazioni, con l'avanzare degli anni tutto ciò sta causando una crescita graduale degli attacchi informatici nel mondo, i quali stanno diventano sempre più frequenti e complessi coinvolgendo sempre più tipologie di vittime e causando danni economici sempre maggiori. Lo sviluppo della criminalità informatica è dovuto anche al modo in cui possono operare gli hacker, infatti lo spazio cibernetico consente loro di poter compiere un attacco i qualsiasi posizione fisica colpendo in modo istantaneo l'obiettivo e in modo quasi del tutto anonimo.
Internet è nato come un' *ungoverned space* ovvero non regolamentato da autorità politiche o internazionali [2], tuttavia a causa di una maggiore pressione in ambito di sicurezza pubblica e a causa di attacchi che in più situazioni hanno coinvolto enti nazionali, diverse nazioni si stanno mobilitando per cercare di offrire più sicurezza online ai propri cittadini. In questo

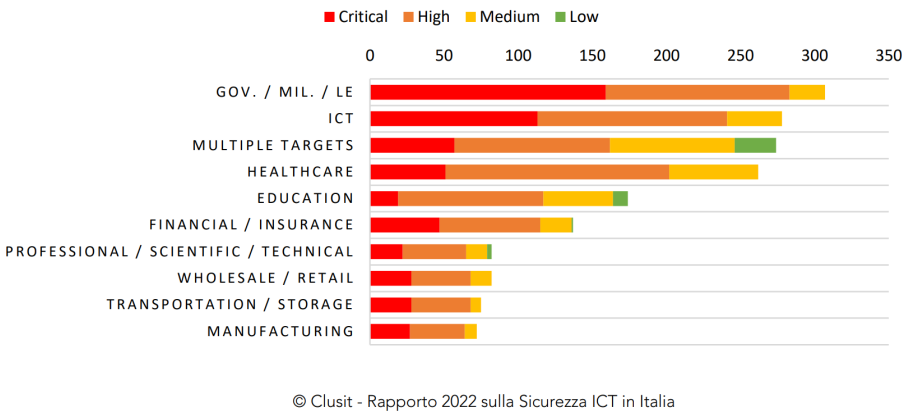

Figura 1.1: Top 10 vittime 2021 in Italia

senso uno degli attacchi hacker che ha fatto molto discutere in Italia è avvenuto il 30 luglio 2021 al CED (centro elaborazione dati) della regione Lazio [3], l'attacco ha mandato in tilt i servizi di privati e aziende, coinvolgendo anche il sistema sanitario della regione dedicato alla vaccinazione contro il COVID-19, gli effetti sono stati molto gravi provocando un mese di interruzione di alcuni servizi tra cui alcuni urbanistici, ma coinvolgendo sopratutto servizi sanitari online. L'attacco informatico subito è definito *ransomware* [4] questa tipologia di attacco si diffonde attraverso un programma dannoso (malware), che ha come scopo quello di bloccare l'accesso ai contenuti presenti sul dispositivo infetto, per poi richiedere un riscatto per renderli di nuovo accessibili, nel caso della regione Lazio dalle ricostruzioni l'infezione è partita da un computer di un dipendente, dal quale i criminali sono riusciti ad accedere ai servizi di un dipendente con privilegi di admin, tramite quest'ultimo sono riusciti ad accedere ai dati per poi criptarli.

Attacchi come quello avvenuto alla regione Lazio mostrano come queste attività criminali possono incidere notevolmente sulla sicurezza delle istituzioni e dei cittadini, questi eventi stanno portando gli enti nazionali e internazionali a mobilitarsi per poter prevenire e combattere questa tipologia di criminalità.

In Italia, negli ultimi anni, il tema della sicurezza informatica sta assumendo una posizione sempre più rilevante, uno degli interventi più importanti è il decreto-legge n. 105 del 2019 il quale è nato con lo scopo di garantire un elevato livello di sicurezza delle reti e dei sistemi informatici delle pubbliche amministrazioni, dei servizi informatici e degli enti e operatori nazionali [5]. Attualmente la sicurezza cibernetica è uno dei progetti principali finanziati dal *PNRR* (Piano nazionale di ripresa e resilienza) [6], a cui sono destinati 620 milioni di euro per il rafforzamento delle infrastrutture legate alla protezione cibernetica in Italia. Anche

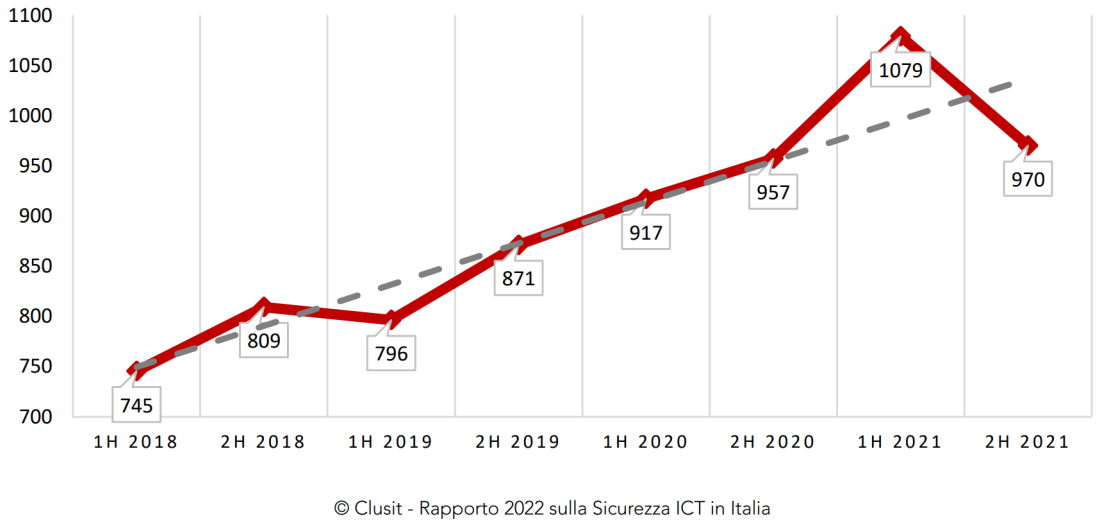

Figura 1.2: Attacchi per semestre dal 2018

l'Europa attraverso la Commissione europea sta spingendo gli Stati membri a rafforzare le proprie difese informatiche, in particolare nel 2016 è stata introdotta la prima misura legislativa per tutta l'Europa in ambito di sicurezza cibernetica, la *Direttiva NIS* [7], che poi, a fronte di una notevole accelerazione della digitalizzazione causata dalla crisi COVID-19, la Commissione europea nel dicembre del 2020 ha proposto la direttiva NIS riveduta (*NIS2*)[8]. Con il trascorrere degli anni il numero degli attacchi informatici sta aumentando, con essa sta crescendo la gravità degli attacchi e la loro complessità, aumentando i costi ad esso associati. Il **Clusit** (Associazione Italiana per la Sicurezza Informatica) [9] periodicamente rilascia report sulla situazione italiana in ambito di sicurezza informatica, nell'ultimo rapporto avvenuto a marzo del 2022 [10] il Clusit ha riportato che nel 2021 il numero di attacchi informatici nel mondo ha registrato un aumento di circa 10% rispetto al 2020, un dato molto preoccupante è l'aumento dell'impatto degli attacchi infatti il 79% di questi ha avuto un impatto elevato. Un importante analisi è stata condotta per il territorio italiano che ci permette di analizzare la situazione in Italia, dalla figura 1.2 possiamo denotare come dal 2018 il numero di attacchi in Italia è cresciuto enormemente, con un picco di 1079 attacchi nel primo semestre del 2021, un dato molto importante da considerare in ambito di sicurezza informatica sono le vittime degli attacchi, infatti gran parte di queste, come si evince dalla figura 1.1, sono enti nazionali e governativi, rispetto al 2020 però le categorie che hanno avuto un aumento considerevole di attacchi nell'ultimo anno sono Transportation/Storage con un aumento del 93,3% e la categoria di News/Multimedia con un aumento del 60,5%.

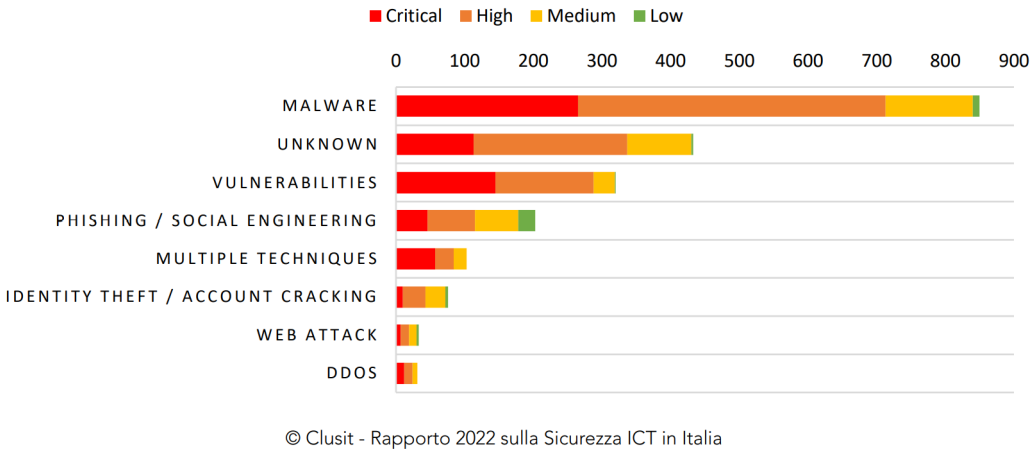

Figura 1.3: Tecniche di attacco 2021

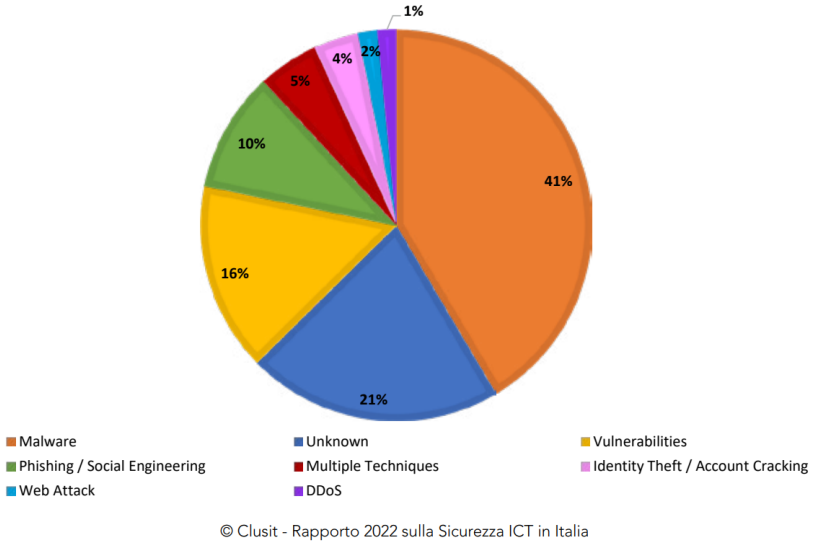

Figura 1.4: Distribuzione tecniche di attacco 2021

Uno dei dati più interessanti anche in termini di prevenzione è il modo attraverso cui gli hacker riescono ad effettuare un attacco informatico, figura 1.3 e figura 1.4 ci mostrano quali sono e in che modo sono distribuiti, da quanto possiamo notare il metodo più diffuso sono i malware con una percentuale del 41%, mentre il 21% è sconosciuto e il 16% degli attacchi avviene sfruttando vulnerabilità, inoltre confrontando i dati con l'anno precedente notiamo un notevole aumento di attacchi avvenuti sfruttando vulnerabilità con un aumento del 60% di quest'ultimi, mentre i malware hanno registrato un aumento del 9.7%, oltre a questi dati notiamo anche un incremento dell'impatto di quest'ultimi, anche in questo caso le vulnerabilità hanno registrato un aumento soprattutto di attacchi critici e alti.

Come mostrato gli aspetti più critici della sicurezza informatica sono lato software, per questo

motivo gran parte degli studi coinvolti in ambito di sicurezza si concentrano sui sistemi software. La sicurezza di un sistema software nasce dalla sua origine, durante la definizione dei requisiti, in particolare durante questa fase sono definite le politiche di sicurezza che specificano ciò che è consentito e ciò che non è consentito dal sistema, più nel dettaglio definiscono la disponibilità del sistema e la definizione per ogni tipologia di utente di cosa può fare e a cosa può accedere e nel caso questi requisiti non vengano rispettati come il sistema deve comportarsi, in modo tale funzionare correttamente in caso di un errore o in caso di un attacco al sistema [11]. Questo mostra come un buon sviluppo del software può in parte contribuire con la creazione di un prodotto software sicuro da possibili attacchi informatici, sfortunatamente, tutto ciò non è sufficiente per evitare la presenza di possibili vulnerabilità software.

1.2 Motivazioni e Obiettivi

Come abbiamo visto la sicurezza informatica è uno dei problemi più rilevanti in campo informatico, che colpisce principalmente persone e istituzioni, attaccando la loro sicurezza e minando profondamente la salute e la privacy degli utenti. Quindi combattere la diffusione di attacchi informatici non solo ci permette di contrastare organizzazioni criminali e truffe, ma anche a salvaguardare la sicurezza nazionale, internazionale e dei singoli cittadini, la lotta contro la criminalità informatica ci permette inoltre di abbassare drasticamente i costi dovuti a danni o a truffe online.
La prevenzione è uno dei metodi più efficaci per abbattere l'aumento di crimini informatici, un'infrastruttura o un prodotto software che offrono buoni sistemi di protezione contro possibili attacchi informatici sono meno vulnerabili e più affidabili, quindi sono più difficili da compromettere, lato software la realizzazione di sistemi sicuri, prevede inevitabilmente un innalzo dei costi di implementazione, sfortunatamente a causa di budget ridotti solitamente per sistemi software medio-piccoli si tende a raggiungere un compromesso tra budget (in termini di denaro e tempi di sviluppo) con requisiti del prodotto tra cui il grado di sicurezza di questo, tutto ciò però inevitabilmente crea un deficit di sicurezza in che rende il sistema software non del tutto sicuro e affidabile.
Il mio lavoro di tesi si pone l'obiettivo di trattare e analizzare la terza categoria di attacchi informatici più diffusi in Italia, ovvero le vulnerabilità, nello specifico il mio studio si concentrerà sulle tecniche di individuazione delle vulnerabilità software attraverso modelli di predizione basati su machine learning. L'obiettivo è minimizzare l'introduzione di vulne-

rabilità in un progetto software e di minimizzare i costi ad esso associati, favorendo anche l'individuazione di vulnerabilità in fase di implementazione, testing e manutenzione di un progetto software, rendendo quest'ultimo più sicuro e affidabile.

1.3 Risultati ottenuti

Nella prima parte di questo studio la mia attenzione è stata posta sullo studio e l'analisi delle vulnerabilità software, come queste possono essere introdotte all'interno di un prodotto software, in che modo possono essere usate dai criminali informatici e che impatto possono avere sulle aziende produttrici e sui loro clienti, in questa prima fase ho analizzato diversi studi fatti in questo ambito, particolare attenzione in questa fase è stata rivolta sui VPM (modelli di predizione di vulnerabilità) e sulle performance ottenute da quest'ultimi in vari ambiti. A seguito di quanto appreso da lavori antecedenti, ho compreso come e in che modo costruire il modello di predizione definendo obiettivi, ambito applicativo e le fasi di implementazione del modello.
Il contesto applicativo scelto sono applicazioni Java e il modello di machine learning ha come obiettivo la predizione di vulnerabilità a livello di singoli file, questo studio ha come ulteriore scopo di indagare sulla correlazione di diverse metriche software con la presenza di vulnerabilità software. Successivamente sono passato alla parte implementativa, in una prima fase la mia attenzione è stata rivolta sui dati e sul dataset di partenza per il modello di machine learning, ho deciso di creare due dataset distinti a partire da due progetti open source di diversa complessità e grandezza, successivamente ho creato il modello di machine learning attraverso l'uso della libreria Python Scikit learn. L'implementazione è avvenuta analizzando e implementando diverse tipologie di classificatori, in modo tale da analizzare le differenze in termini di prestazioni e risultati in questo ambito utilizzando diversi classificatori. Analizzando i risultati ottenuti ho notato una buona accuratezza in quasi tutti i modelli con una percentuale che varia tra l'80% e 90%, mentre ho riscontrato per tutti i modelli analizzati un livello di recall più alto rispetto alla precision, il classificatori con indici prestazionali più alti sono stati il **Decision tree** e il **Random forest** viceversa il classificatore con risultati più bassi è stato **Naive Bayes**.

1.4 Struttura della tesi

Nel capitolo 2 sono presentate e analizzate le vulnerabilità in campo informatico, con una particolare attenzione rivolta alle vulnerabilità software, successivamente nella seconda parte di questo capitolo sono descritti e analizzati diversi studi fatti in passato per la creazione e l'implementazione dei VPM (modelli di predizione di vulnerabilità) e i risultati ottenuti da quest'ultimi, nell'ultima parte di questo capitolo sono presentati e analizzati diversi dataset utilizzati in passato per la creazione dei VPM.
Nella prima fase del capitolo 3 è presente la definizione degli obiettivi posti per la realizzazione del modello di predizione e il contesto applicativo dove quest'ultimo dovrà operare, successivamente c'è la descrizione del lavoro di estrazione dei dati e la costruzione del dataset, e infine il processo di analisi, implementazione e validazione del modello di predizione.
Nel capitolo 4 sono descritti e analizzati i risultati ottenuti dal modello di machine learning realizzato in questo studio.
Nel capitolo 5 sono descritte le conclusioni e possibili sviluppi futuri legati a questo studio.

CAPITOLO 2

Background e Stato dell'arte

2.1 Vulnerabilità software

Le vulnerabilità sono errori o componenti software dove le misure di sicurezza sono assenti o compromesse, questi spesso rendono l'intero sistema vulnerabile e possono essere sfruttati da malintenzionati per compiere possibili attacchi informatici [12]. Solitamente non sono le vulnerabilità in sé ad essere il problema e a danneggiare un sistema informatico, ma esse possono essere sfruttate per diffondere infezioni o possono essere il mezzo attraverso cui un malintenzionato può manipolare il sistema. Una vulnerabilità software diventa pericolosa quando questa viene scoperta infatti l'attaccante può sfruttare una vulnerabilità in modo tale da ottenere privilegi maggiori di quelli a lui concessi cosicché accedere a file o risorse a cui non potrebbe accedere, le vulnerabilità quindi costituiscono un'arma molto potente per gli hacker in quanto permettono di aggirare i sistemi di protezione rendendoli del tutto inutili e a consentire lo svolgimento, sul sistema attaccato, di operazioni non autorizzate. Possiamo classificare le vulnerabilità informatiche in tre categorie:

- **Vulnerabilità software [12]:** vengono definite anche bug software e sono malfunzionamenti legati a difetti di progettazione, codifica, installazione e configurazione del software. Tipicamente vengono introdotti tramite difetti nel codice o a causa di controlli non adeguati dei dati ricevuti in input dall'applicazione. Questa categoria di vulnerabilità è la più diffusa;

- **Vulnerabilità dei protocolli [12]:** si presentano quando protocolli di comunicazione hanno lacune di sicurezza nel sistema di comunicazione, un esempio è la comunicazione in chiaro o non crittografata che permette a possibili malintenzionati di intercettare le informazioni scambiate;

- **Vulnerabilità hardware [13]:** quando in apparati hardware qualsiasi elemento può causare un pericolo al corretto funzionamento di una macchina tecnologica compromettendone la sicurezza, possiamo fare rifermento all'introduzione di umidità, polvere o qualsiasi cosa all'interno della macchina.

Le vulnerabilità software sono tra le più diffuse, e sono anche il modo principale attraverso cui gli hacker compiono un attacco informatico, solitamente questo tipo di vulnerabilità viene introdotto durante le fasi del ciclo di vita del software in particolare durante le fasi di progettazione, programmazione, testing e installazione.
Di notevole importanza sono le vulnerabilità zero-day [14] queste sono vulnerabilità software scoperte dagli aggressori prima che il produttore del software ne sia venuto a conoscenza, ciò rende il software vulnerabile ad attacchi informatici finché non viene rilasciata una patch che corregge tale vulnerabilità. Un' esempio molto discusso di vulnerabilità zero-day è stata scoperta a luglio del 2020 da parte di Opatch [15] [16] sul software di videoconferenze *Zoom*, questo software rendeva vulnerabili a possibili attacchi informatici tutti i sistemi operativi Windows 7 e tutte le versioni precedenti. La vulnerabilità riscontrata fu classificata come Remote Code Execution (RCE) questa permetteva l'esecuzione di codice remoto sulla macchina attaccata e durante l'attacco l'utente non veniva avvisato, fortunatamente la vulnerabilità venne tempestivamente risolta da parte di Zoom dopo solo un giorno dalla sua segnalazione.
Uno dei progetti open-source più importanti per la sicurezza web è l'OWASP [17] (Open Web Application Security Project) questo ha come scopo quello di fornire agli sviluppatori articoli, documenti e linee guida per l'implementazione e la realizzazione di sistemi software sicuri. Tra le iniziative più importanti intraprese dall'OWASP è la top 10 web application security risk, nella quale elenca le vulnerabilità software più comuni e rischiose che solitamente vengono introdotte nei sistemi software, inoltre per ognuna di esse fornisce una descrizione dei possibili rischi e di come evitare queste vulnerabilità.
Esistono diversi tipi di vulnerabilità software i più comuni sono:

- *Buffer overflow [18]:* avviene quando per errore o malizia vengono inviati dati in input ad un buffer più grandi della capienza del buffer stesso, i dati eccessivi sovrascrivono

le variabili interne o lo stack della memoria, questo errore compromette il programma, il quale può avere un comportamento imprevisto o bloccarsi del tutto, se vengono immessi dati malevoli essi possono prendere il controllo del programma o peggio il controllo del intero computer. Non tutti i linguaggi di programmazione possono riscontrare questa vulnerabilità;

- *Injection [19]:* questo difetto permette agli aggressori l'iniezione di codice tramite chiamate di sistema, queste solitamente vengono eseguite utilizzando programmi esterni tramite comandi della shell. Le iniezioni al database in particolare le iniezioni SQL injection solo le comuni e pericolose;

- *Broken Authentication [19]:* questo tipo di vulnerabilità si verifica quando un malintenzionato riesce ad utilizzare modi diversi per entrare nel account di qualcun altro. Questo tipo di vulnerabilità colpisce principalmente le applicazioni web, le cause principali sono la mancanza di verifiche o timeout in una sessione. Un altro modo è quando l'hacker riesce ad accedere al database delle password e sfruttando altre vulnerabilità riesce a convertire la codifica di queste permettendogli di visualizzare le password di tutti gli utenti;

- *Broken Access Control:* questa vulnerabilità porta alla mancanza di controllo su chi ha accesso per leggere e modificare i dati. Nella maggior parte delle volte questa vulnerabilità porta ad accesso, utilizzo, modifica o divulgazione di dati non autorizzati;

- *Criptographic failure [19]:* questo tipo di vulnerabilità espone dati sensibili come password, informazioni personali, numeri di carte di credito etc.., esso solitamente è derivato da usi scorretti della crittografia, da algoritmi crittografici deboli o da trasmissioni di dati in chiaro su protocolli di comunicazione come FTP o HTTP;

- *Software and data Integrity failures [19]:* è riferito a codice che non protegge da violazioni di integrità. Solitamente questo tipo di vulnerabilità viene introdotta quando un' applicazione si basa su plug-in o librerie che provengono da fonti non attendibili, un altro tipico esempio è quando applicazioni che offrono servizio di aggiornamento automatico, alcuni di essi non effettuano opportuni controlli sul integrità dei dati sui nuovi aggiornamenti, applicandoli ad applicazioni precedentemente attendibili. Questo codice può introdurre codice dannoso che potrebbe compromettere l'integrità del sistema.

Gli attacchi informatici più comuni sono definiti crimini di dati [20], dove gli attaccanti hanno come obiettivo quello di rubare, manipolare e intercettare dati per trarne profitto, un'altra tipologia molto diffusa di crimine informatico è la disseminazione di virus [20], i virus sono software dannosi che hanno come obiettivo quello di distruggere il sistema della vittima. Tutti questi crimini hanno un' impatto sulle aziende e sugli utenti molto forte soprattutto in termini economici ma anche di sicurezza e privacy [20].

Nonostante tutti gli sforzi fatti per la prevenzione e per l'individuazione delle vulnerabilità software, spesso queste tecniche sono molto costose, il che rende la prevenzione e la risoluzione di vulnerabilità metodi poco efficienti, per questo motivo nell'arco degli anni diversi studi si sono concentrati sull'individuazione delle vulnerabilità attraverso tecniche di predizione, in particolare attraverso l'uso di machine learning.

2.2 Modelli di apprendimento per predizione di vulnerabilità

Come abbiamo visto le vulnerabilità software sono una problematica molto importante da tenere in considerazione durante lo sviluppo e ciclo di vita di sistema software, sfortunatamente a causa delle risorse limitate non è possibile condurre un controllo dettagliato in fase di sviluppo o di testing per trovare e risolvere vulnerabilità software, per questo motivo gli sviluppatori negli ultimi anni si sono concentrati sulla creazione di strumenti in grado di automatizzare la predizione di vulnerabilità [21], tramite l'intelligenza artificiale, in particolare nell'ambito del machine learning sono stati compiuti molti studi per cercare di risolvere questo problema, in questo campo un'attenzione particolare è stata posta sui modelli di predizione di vulnerabilità software (VPM).
Nel machine learning la scelta delle "software feature" gioca un ruolo molto importante per il corretto funzionamento e per migliorare gli indici di prestazione, per cui uno dei primi obiettivi dei VPM consiste nell'aumentare l'accuratezza della previsione limitando i costi ed estraendo attributi software appropriati [22].
Una prima categoria usata in vari algoritmi sono i predittori basati sulle metriche software, queste rappresentano tramite valori numerici caratteristiche di un software, questa tipologia di metriche considerano che al crescere della complessità e della grandezza di un sistema software esso diventa potenzialmente sempre più esposto a vulnerabilità software, per questo motivo solitamente vengono prese in considerazione metriche che considerano la complessità del codice tra i principali abbiamo: numero di funzioni, linee di codice (LOC), numero di chiamate a metodi o funzioni esterne, numero di chiamate a metodi o funzioni interni ed

esterni, numero di nidificazione e numero di file (Fan-in e Fan-out) [23].
Uno dei primi studi è stato effettuato da Zimmerman et al. [24] in questo studio sono stati effettuati una serie di test empirici su Windows Vista in cui hanno provato l'efficienza di varie metriche software classiche come metriche di complessità, code churn e misure di dipendenza, hanno riscontrato un'alta precisione ma un basso recall. Similmente Shin et al. [25] crearono una serie di modelli divisi in tre categorie: complessità, code churn e sulle attività degli sviluppatori, gli studi furono effettuati su Mozilla Firefox web browser e the Red Hat Enterprise Linux kernel, in totale hanno ottenuto e testato 28 metriche e 24 di queste sono riuscite a discriminare vulnerabilità software. Neuahus et al. [26] nel contesto di Morzilla Firefox crearono un nuovo strumento per la predizione di vulnerabilità chiamato "Vulture", il suo funzionamento prevede l'associazione di vulnerabilità passate a componenti software tramite un database di vulnerabilità, il predittore risultante prevede vulnerabilità dei nuovi componenti in base agli import o alle funzioni chiamate, Vulture riscontrò un'alta precisione 70% ma un basso recall 45%. Nguyen et al. [27] proposero un nuova metrica basata sui grafi delle dipendenze delle componenti di un sistema software (CDG), questi grafi sono basati sulle relazioni tra gli elementi software come classi, funzioni o variabili, ciò rende i grafi estraibili tramite un'analisi del codice sorgente o dai dettagli delle specifiche di design, per questo motivo questa metrica è possibile usarla sia in fase di sviluppo che in fase di design. Questo modello fu sperimentato su JavaScript Engine of Firefox ed ottennero risultati migliori confrontandoli con i modelli basati su metriche di complessità, ottenendo un'alta accuratezza di circa 84% e un alto recall di circa 60%.
Scandariato et al. [28] furono i primi a proporre un modello di predizione delle vulnerabilità basato sul text mining, usarono la tecnica del "bags of words", dove hanno estratto un insieme di termini più frequenti nel codice sorgente Java e posti in un dataset vennero usati come features per la predizione del algoritmo usato, lo studio fu condotto su 20 applicazioni android, ed ottennero una precisione e un recall di circa l'80%, Walden et al. tramite uno studio [23] cercarono di fare confronto tra le classiche metriche software e il text mining. Il test fu effettuato nelle stesse condizioni per entrambe le metriche, il dataset era composto da 223 vulnerabilità riscontrate in applicazioni web, fu usato il classificatore Random Forest e furono testati su tre progetti. Il text mining riscontrò una maggiore precisione e un maggiore recall rispetto alle metriche software, tuttavia riscontrarono pessimi risultati durante la predizione su un progetto usando un predittore sviluppato su un altro progetto, in aggiunta si riscontrò che in entrambi i modelli all'aumento della grandezza del prodotto si verificò una perdita di precisione.

Come mostrato da Theisen e Williams [29] la combinazione di feature da più modelli non necessariamente aumenta l'accuratezza del predittore anzi in alcuni casi si ha un degrado di prestazioni quindi il migliore approccio per migliorare le prestazioni di un VPM consiste nel trovare il miglior classificatore e le migliori feature da dargli in input.

2.3 Dataset di apprendimento per predizione di vulnerabilità

Uno degli elementi più importanti di un machine learning è un dataset, infatti quest'ultimo è il uno dei punti cardini per l'implementazione e il corretto funzionamento di un machine learning, per questo motivo la scarsa qualità di un dataset può influire negativamente sulla qualità della predizione prodotta.
I dataset presenti attualmente possono essere classificati in base al loro target di prodotti [30], la prima tipologia sono i dataset multi-vendor, questi hanno la caratteristica di includere un largo numero di vulnerabilità, riscontrabili in qualsiasi tipo di ambiente software, tra i più importanti ci sono:

- **CVE** (Common Vulnerabilities Exposure) [31]: questo dataset è pubblico ed è gestito da Mitre Corporation, è nato con lo scopo di identificare e catalogare in un database pubblico vulnerabilità software. Ogni vulnerabilità del dataset ha un ID, una piccola descrizione e dei riferimenti. La pubblicazione di nuove vulnerabilità può avvenire direttamente dal MITRE o attraverso i CNA (CVE Numbering Authority) che rappresentano i principali fornitori IT o enti di ricerca (Microsoft, Red Hat, Oracle, etc.);

- **NVD** (National Vulnerability Database) [32]: è un progetto del dipartimento di sicurezza nazionale degli Stati Uniti ed è stato creato con lo scopo di aiutare persone e aziende nel processo di automatizzazione delle vulnerabilità software. I dati sono rappresentati tramite il Security Content Automation Protocol (SCAP), questo protocollo associa ad ogni vulnerabilità presente nel database un gran numero di dati e informazioni utili per la gestione automatizzata delle vulnerabilità, tra i più importanti troviamo il CVSS (Common Vulnerability Scoring System) che assegna un punteggio alla gravità della vulnerabilità esaminata, il CWE (Common Weakness Enumeration) che descrive il tipo di vulnerabilità e il CPE (Common Platform Enumeration) che descrive la classe di applicazioni e sistemi operativi dove la vulnerabilità in questione è riscontrabile. Le vulnerabilità vengono aggiunte al NVD a partire da CVE, infatti queste vengono analizzate e integrate con il database;

- **OSVDB** (Open Source Vulnerability Database): è un database indipendente e open-source nato con lo scopo di descrivere in modo accurato le vulnerabilità software.

La seconda categoria sono i dataset vendor, questi sono creati appositamente per un applicazione o un sistema operativo, in questi troviamo tutte le vulnerabilità riscontrabili in quella tipologia di prodotti, tra i più importanti troviamo:

- **MFSA** (Mozilla Foundation Security Advisories) [33]: sono tutti i report riscontrati nei prodotti Mozilla, ad ogni report vengono elencati tutte le vulnerabilità riscontrate e corrette, per ognuna di queste viene assegnato un valore di impatto e una descrizione;

- **Bugzilla**: esistono due istanze di Bugzilla, RedHat Linux Bugzilla [34] che comprende tutte le vulnerabilità del sistema operativo RedHat Linux, Mozilla Bugzilla [35] invece è un database che comprende tutte le vulnerabilità presenti in tutti i prodotti Mozilla.

CAPITOLO 3

Metodologia di analisi

3.1 Obiettivi dello studio

Questo studio si pone l'obiettivo di indagare su come e in che modo un predittore basato su machine learning può individuare la presenza di vulnerabilità software in un file sorgente, analizzando su come integrare il predittore durante la creazione del prodotto software, con lo scopo di facilitare l'individuazione di possibili criticità di sicurezza durante lo sviluppo del software. Entrando più nel dettaglio lo studio si concentrerà sull'analisi di diverse metriche software, indagando come queste possono influire sulla presenza di possibili vulnerabilità software e quali sono le più caratterizzanti tra queste. L'implementazione è stata guidata seguendo modelli di machine learning creati in studi precedenti fatti in altri contesti applicativi, che sono stati descritti nel capitolo 2, cercando di adattare quanto più possibile il nuovo modello creato al mio contesto, in modo tale da migliorarne le prestazioni.

3.2 Contesto dello studio

L'obiettivo di questo studio è la creazione di un modello di machine learning per la predizione di vulnerabilità software in singoli file per applicazioni Java, questa scelta è stata guidata dall'importanza che il linguaggio Java ricopre nel campo informatico, infatti con il passare degli anni le applicazioni scritte in questo linguaggio sono sempre più numerose e complesse, ricoprendo un ruolo sempre più influente nella nostra società, per cui lo sviluppo

di un modello di predizione in questo ambito aiuterebbe un gran numero di aziende e di sviluppatori, inoltre ho deciso di analizzare i singoli file in modo tale da avere una predizione che desse un'indicazione precisa su dove individuare la vulnerabilità presente nel prodotto software preso in considerazione.
Per il mio studio ho deciso considerare due progetti Java open-sorce, i progetti scelti sono Cloud Foundry [36] e Spring Framework [37]. Come mostrato dalla tabella 3.1 i due progetti hanno differenti grandezze e il progetto Spring Framework ha una grandezza e una complessità maggiore, in totale nei due progetti abbiamo 7 723 file tra questi ho estratto 144 file vulnerabili. Per l'estrazione dei file vulnerabili ho considerato il dataset [38] creato nel 2019, il quale è stato creato con l'intenzione di collezionare per ogni progetto preso in considerazione tutte le vulnerabilità riscontrate durante lo sviluppo e la manutenzione dei seguenti prodotti software, tra questi sono stati analizzati anche i progetti presenti in questo studio.
L'intera implementazione del modello è stata sviluppata e scritta in Python attraverso il tool online Google Colab, tutto il progetto compreso di dataset e Jupiter Notebook è presente su GitHub ed è accessibile attraverso il seguente link `https://github.com/larsyx/Vulnerability-Detection`. Per l'implementazione e l'analisi del modello di machine learning ho usato la libreria Python open-source *Scikit-learn [39]*, che mi ha permesso di implementare in modo veloce e completo l'intero modello di predizione.

Progetto	Commits	File	File vulnerabili
Cloud Foundry	9 844	734	58
Spring Framework	24 903	6 989	86
	34 747	**7 723**	**144**

Tabella 3.1: Statistiche dei progetti presi in considerazione nello studio

3.3 Estrazione dei dati

In questa prima fase ho preparato e analizzato i dati che poi mi serviranno per il corretto funzionamento del modello di machine learning. La creazione del dataset è avvenuta attraverso il seguente dataset [38], da quest'ultimo ho estratto i file vulnerabili che sono stati risolti in passato per entrambi i progetti, ogni dato di questo dataset era composto da: progetto github, commit-ID in cui la vulnerabilità è stata risolta e il CVE-ID [31] della vulnerabilità considerata, per l'estrazione dei file ho selezionato tutti i record presenti nel dataset per il progetti in questione e per ogni record ho estratto il file avente una vulnerabilità software

attraverso il commit-ID. Dopo ciò ho ricavato un totale di 144 file vulnerabili dove 56 di questi nel progetto Cloud Foundry mentre 86 in Spring Framework.
Il modello implementato dovrà predire la presenza di vulnerabilità software attraverso diverse metriche software, per cui successivamente ho deciso di porre la mia attenzione sulla selezione e l'individuazione di diverse metriche software che possono avere un impatto rilevante per la presenza di vulnerabilità in un file sorgente. In una prima fase ho scelto le metriche software che andranno a caratterizzare il dataset creato, nello specifico ho deciso di prendere in considerazione 15 metriche software che sono riassunte nelle Tabelle 3.3 e 3.2, il motivo per cui ho deciso usare queste metriche è derivato dalla natura stessa del problema, infatti ho cercato di individuare tra le diverse metriche software, sulla base degli studi precedentemente fatti e analizzati nel capitolo 2, quelle più adatte al mio contesto.
Gran parte delle variabili indipendenti scelte mostrate nella tabella 3.3 sono le più classiche metriche software molte di queste estraibili facilmente dal codice sorgente, tra queste una delle metriche più interessanti per la valutazione di un file sorgente è la "*Cyclomatic complexity*", quest'ultima è stata sviluppata con l'intenzione di stimare attraverso un numero il livello di complessità di un file, l'estrazione della complessità ciclomatica avviene individuando il numero di cammini linearmente indipendenti presenti in un file il risultato della somma di quest'ultimi indica il livello di complessità del file sorgente.

NOME	DESCRIZIONE
HalsteadVocabulary	Il totale tra tutti gli operatori unici e il numero di operandi unici.
HalsteadLenght	Il totale tra il numero di occorrenze degli operandi e il numero di occorrenze degli operatori.
HalsteadVolume	Rappresenta le dimensioni dello spazio necessario per ordinare un programma, il volume è proporzionale alla lunghezza del programma.

Tabella 3.2: Metriche di Halstead

Tra le più interessanti metriche software scelte troviamo quelle create da Maurice Howard Halstead descritte nella tabella 3.2, queste metriche sono state sviluppate con lo scopo di misurare la complessità di un programma software senza la sua esecuzione, infatti queste sono state create con l'idea che le metriche software dovrebbero riflettere l'implementazione di algoritmi che compongono il programma. L'idea di base per l'estrazione delle metriche

di Halstead sta nella suddivisione del codice sorgente in una sequenza di token, i token successivamente sono classificati e contati come *operatori* e *operandi*.

L'estrazione delle metriche software per la creazione del dataset è avvenuto attraverso il software "*SwAnalytics*" creato dal SeSa Lab [40], quest'ultimo mi ha permesso l'estrazione in formato csv di un intero progetto software fruibile su GitHub, il seguente programma mi ha dato anche la possibilità di scegliere il commit-ID del progetto di cui fare l'estrazione delle metriche. Per cui ho preso in considerazione l'ultimo commit effettuato dai progetti software di cui ho preso in considerazione, ho estratto i file contenti i risultati ed ho considerato quest'ultimi come file non vulnerabili. Successivamente ad ogni commit estratto precedentemente dal dataset [38], il quale mi dava conoscenza delle vulnerabilità presenti all'interno del progetto, ho estratto le metriche attraverso il commit-ID e dal file risultante ho estratto i file vulnerabili, che sono stati inseriti nel nuovo dataset costruito.

3.4 Analisi dei dati

In questa fase del mio studio il lavoro eseguito è stato svolto per la preparazione dei dati e per l'implementazione e validazione del modello di machine learning. Prima di iniziare con queste fasi ho analizzato il dataset e il problema in questione e ho notato che il dataset usato per l'implementazione del modello di predizione contiene dati strutturati.
Prima di procedere con la fase di preparazione dei dati ho suddiviso il dataset iniziale in *training set* e *test set*.

3.4.1 Preparazione dei dati

Durante questa fase il mio obiettivo è stato di preparare i dati per la successiva fase di addestramento, infatti una maggiore qualità dei dati in fase di addestramento contribuisce in modo significativo alla costruzione di un modello di machine learning che operi in modo corretto e con un alto livello di precisione, questa è una delle fasi più cruciali per ottenere un modello di predizione efficiente e che operi in modo corretto successivamente nella fase di rilascio. Alcune delle seguenti operazioni sono state effettuate esclusivamente sul training set in modo tale da evitare fenomeni di *data leakage*, questo problema è causato da diversi fattori, la causa principale è l'uso durante il processo di addestramento di informazioni che non dovrebbero essere disponibili al momento della predizione [41], un modello di predizione affetto da data leakage rende il modello capace di lavorare in modo accurato in fase di addestramento ma non in fase di rilascio.

NOME	NOME ESTESO	DESCRIZIONE
LOC	Lines of Code	Conteggio numero di linee di codice e di commenti.
LCOM	Lack of Cohesion of Methods	Descrive la mancanza di coesione tra i metodi della classe attraverso un numero, la numerazione avviene in modo crescente quindi un numero molto basso indica che la classe ha un livello di coesione molto alto, viceversa per un numero molto alto.
CBO	Coupling between Objects	Indica il numero di classi che sono legate a una particolare classe.
ELOC	Effective Lines of Code	Indica il numero effettivo di linee di codice togliendo commenti e spazi vuoti.
McCabeMetric	Cyclomatic complexity	Questa metrica stima il livello di complessità di una classe, la stima avviene analizzando cammini linearmente indipendenti di un file sorgente.
RFC	Response For Class	Indica il numero di metodi che possono essere eseguiti quando un oggetto di quella classe riceve un messaggio.
MPC	Message Passing Coupling	Indica il numero di messaggi che passano tra gli oggetti di una classe. Un valore alto indica un forte accoppiamento tra più classi.
DAC	Data Abstracting Coupling	Indica il numero di attributi in una classe che appartengono ad altre classi.
DAC2	Data Abstracting Coupling 2	Indica il numero di classi di cui gli attributi di una classe appartengono.
WMC	Weighted Methods for Class	Questa metrica indica la somma della complessità di tutti i metodi di una classe.
NOA	Number of attributes	Indica il numero di attributi di una classe.
NOPA	Number of public attributes	Indica il numero di attributi pubblici presenti in una classe.

Tabella 3.3: Metriche software

Pulizia dei dati

La pulizia dei dati è la prima fase per la preparazione dei dati e ha l'obiettivo di rimuovere dal dataset dati superflui che non saranno utili per la predizione, e di rimuovere dati mancanti o nulli, in modo tale da dare al modello di predizione dati completi e non generino problemi di data leakage.
Considerando il dataset costruito precedentemente ho deciso di rimuovere le colonne che indicano il commit-ID del progetto e il nome della classe che presentava una vulnerabilità, il motivo è legato alla natura dei dati infatti quest'ultimi non hanno nessuna correlazione con la presenza di vulnerabilità e potrebbero compromettere il corretto funzionamento del modello di predizione, inoltre durante la predizione, dopo il suo rilascio, il modello non avrà a disposizione questi dati. Successivamente ho analizzato il dataset e ho riscontrato che il dataset non presenta di dati mancanti o nulli.

Normalizzazione delle feature

Lo scopo di questa fase è quello di normalizzare tutte le variabili indipendenti del nostro dataset in modo tale che il modello di machine learning predirrà attraverso l'uso di variabili che hanno la stessa scala di valori, la normalizzazione permette di evitare che ordini di grandezza diversi vadano ad avere un impatto sul modello.
Analizzando il mio dataset ho notato che tutte le metriche software presenti nel mio dataset sono variabili numeriche e non categoriche e con scale molto diverse tra loro, quindi ho deciso di normalizzarle tutte, con un scala che va da 0 a 100 per tutte le variabili, a seguito di alcune analisi ho deciso che il processo di normalizzazione dei dati avvenga attraverso la *min-max normalization*, questa tecnica permette di normalizzare i dati in modo veloce ed efficiente a partire dai valori di massimo e di minimo registrati in una determinata caratteristica.

Selezione delle feature

Questa è una delle fasi più importanti dell'intero processo che caratterizzerà fortemente le prestazioni del modello di machine learning, nello specifico durante questa fase si dovranno definire quali e quante variabili indipendenti (feature) saranno utilizzate dal modello di predizione durante la sua esecuzione.
Il modo attraverso cui ho deciso di effettuare la selezione delle variabili è la *feature extraction* ovvero attraverso la generazione automatica delle feature, che avviene attraverso l'uso di diversi algoritmi, Scikit-learn [39] mette a disposizione diversi algoritmi per la feature

extraction, in questo studio ho deciso di utilizzare la funzione *SelectKBest* che mi ha permesso di selezionare le migliori feature del dataset. La decisione sul numero di feature che il modello dovrà utilizzare per la predizione è stato scelto empiricamente effettuando diversi test in fase di modellazione e validazione, alla fine di questo processo ho deciso di selezionare 7 feature, le feature scelte nella versione finale sono: *LCOM*, *CBO*, *McCabeMetric*, *WMC*, *HalsteadVocabulary*, *HalsteadLenght* e *HalsteadVolume*.

Bilanciamento dei dati

Questa è l'ultima fase per la preparazione dei dati, in quest'ultima parte si decide se e in che modo bilanciare il dataset, esistono due strategie di bilanciamento *undersampling* che consiste nel eliminare casualmente istanze della classe di maggioranza e *oversampling* che consiste nella generazione di istanze delle classe di minoranza.
Analizzando il mio dataset e da come si evince dalla tabella 3.1 il dataset presenta un forte sbilanciamento verso le classi non vulnerabili, inoltre nel mio dataset ho riscontrato che non ho a disposizione un elevato numero di istanze, per cui la rimozione di ulteriori istanze porterebbe il modello a lavorare su un numero molto ridotto di dati, sulla base di queste osservazioni ho deciso di generare nel mio dataset istanze di classi vulnerabili in modo tale da bilanciarlo, per la generazione delle istanze ho deciso di usare la funzione *SMOTE* presente nella libreria di Scikit-learn.

3.4.2 Modellazione dei dati

Lo scopo principale di questa fase consiste nella definizione del problema di predizione e nella scelta degli algoritmi che saranno utilizzati, successivamente si passa alla fase di addestramento e nella scelta dei parametri degli algoritmi scelti in modo tale da migliorare le prestazioni del modello di predizione.
Analizzando l'obiettivo del mio modello di predizione ho compreso che il problema è un classificabile come un problema di classificazione, in quanto il predittore dovrà stabilire se un'istanza di una classe è classificabile come file vulnerabile o non vulnerabile. Per l'implementazione ho deciso di scegliere diversi classificatori tra i più comuni, che sono elencati e descritti nella tabella 3.4, lo scopo è di verificare empiricamente le loro prestazioni e valutare quale classificatore ottiene una maggiore precisione in questo ambito. Successivamente in base ai risultati ottenuti da ogni classificatore singolarmente ho definito i parametri per la creazione dei classificatori.

NOME	DESCRIZIONE
Naive Bayes	Questo classificatore fa parte della famiglia dei classificatori probabilistici, e come tale basa la sua predizione sulla applicazione del teorema di Bayes.
Decision Tree	Questo classificatore crea, attraverso il valore di *information gain* per ogni feature, un albero decisionale dal quale poi baserà la sua predizione.
Random Forest	Questo classificatore è un metodo di classificazione *ensemble*, il suo funzionamento consiste nella creazione di diversi alberi decisionali da cui estrae la sua predizione.
Support Vector Machine	Questo classificatore utilizza i vettori di supporto per la creazione di uno o più iperpiani che dividono lo spazio di più dimensioni, il quale sarà utilizzato per la predizione.
AdaBoost	È uno dei classificatori *ensemble*, il suo funzionamento prevede la costruzione di una lista di classificatori aventi ognuno un peso in base alle sue prestazioni, infine la predizione avviene considerando tutti i classificatori in base al loro peso.

Tabella 3.4: Classificatori utilizzati per la creazione del modello di predizione

3.4.3 Validazione del modello

Questa è l'ultima fase del intero processo di creazione del modello di machine learning, l'obiettivo consiste nella verifica del corretto funzionamento del modello di predizione creato, analizzando e verificando che rispetti tutti gli obiettivi posti precedentemente, nel caso che questi non sono rispettati si torna alla fase di modellazione per correggere e migliorare il modello, in questa fase inoltre vengono analizzati, attraverso l'uso di diverse metriche, le prestazioni raggiunte dal predittore.

La strategia di validazione scelta prevedeva la divisione del dataset in training set e test set con un rapporto rispettivo di 67% e 33%, il training set è stato usato esclusivamente per l'addestramento del modello, mentre il test set è stato usato esclusivamente per la validazione del modello, il motivo legato a questa suddivisione sta nella natura del dataset, siccome il mio dataset di partenza non ha un numero di istanze particolarmente corposo, questa proporzione mi permette di dare in input alla fase di addestramento un numero adeguato di istanze. Da questa fase ho appreso come i diversi classificatori si comportano e quali di

questi ha registrato prestazioni più precise e accurate, nello specifico i classificatori che ha registrato indici prestazionali più interessanti sono il *Decision Tree* e il *Random Forest* mentre i risultati più bassi sono stati registrati dal *Naive Bayes*.
La scelta delle metriche di valutazione è avvenuta valutando, tra quelle esistenti, quelle più utili e complete che mi permettono di avere una visione completa e precisa delle prestazioni raggiunte dai diversi modelli implementati, per cui tra le diverse metriche di valutazione ho deciso di scegliere per la validazione le seguenti metriche:

- *recall*: questa metrica indica quante istanze positive nell'intero dataset il classificatore può determinare, la sua formula matematica è $\frac{veropositivo}{veropositivo+falsopositivo}$;
- *precision*: questa metrica indica quanti errori ci saranno nella lista delle predizioni fatte dal classificatore, la sua formula matematica è $\frac{veropositivo}{veropositivo+falsonegativo}$;
- *accuratezza*: questa metrica indica il numero totale di predizioni corrette, la sua formula matematica è $\frac{veropositivo+veronegativo}{veropositivo+veronegativo+falsopositivo+falsonegativo}$;
- *f-measure*: questa metrica è ottenuta attraverso la media armonica tra precision e recall, ed è una misura che indica il livello di accuratezza del classificatore.

Le prestazioni ottenute dai diversi classificatori sono descritte e analizzate nel capitolo 4.

CAPITOLO 4

Analisi dei risultati

In questo capitolo saranno descritti i risultati ottenuti dall'implementenzione dei vari modelli di machine learning realizzati. Una delle prime metriche prese in considerazione è l'accuratezza ottenuta dai vari modelli implementati, quest'ultima da come si evince dalla figura 4.1 ha registrato risultati molto alti per tutti i modelli implementati, nonostante ciò il classificatore che registrato una maggiore accuratezza è stato il Random forest con un risultato di 0,89, mentre il risultato peggiore l'ha ottenuto il classificatore SVM con un risultato di 0,75.

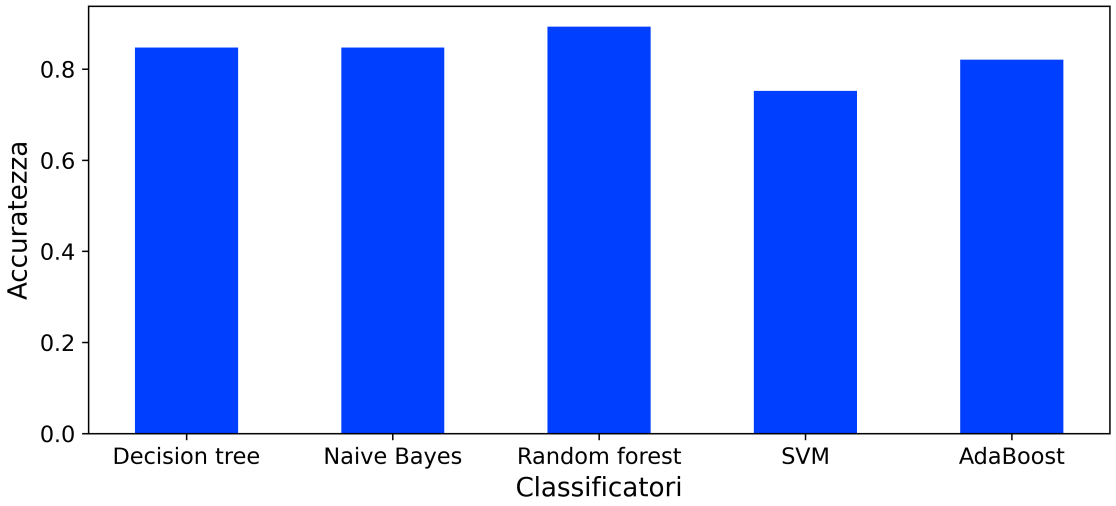

Figura 4.1: Accuratezza classificatori

Successivamente ho preso in considerazione Precision e recall, quest'ultime mi danno la possibilità di analizzare più informazioni rispetto all'accuratezza, e inoltre mi permettono di

avere una visione più completa delle prestazioni dei modelli implementati. Dalla tabella 4.1 possiamo notare che in generale tutti i modelli hanno ottenuto risultati molto buoni per la predizione di file non vulnerabili, ma risultati poco precisi per la predizione di file vulnerabili, tra i diversi dati notiamo che il livello di precisione più alto raggiunto è stato di 0,34 per i file vulnerabili e 0,98 per i file non vulnerabili risultati ottenuti entrambi dal classificatore Random forest, mentre il livello di precisione più basso è stato ottenuto dal classificatore Naive Bayes con 0,08 per i file vulnerabili e 0,95 per i file non vulnerabili.

PREDIZIONE	PRECISION	RECALL
Decision tree		
File vulnerabili	0,26	0,71
File non vulnerabili	0,98	0,86
Naive Bayes		
File vulnerabili	0,08	0,71
File non vulnerabili	0,95	0,43
Random forest		
File vulnerabili	0,34	0,71
File non vulnerabili	0,98	0,91
SVM		
File vulnerabili	0,14	0,53
File non vulnerabili	0,96	0,77
AdaBoost		
File vulnerabili	0,20	0,59
File non vulnerabili	0,97	0,84

Tabella 4.1: Precision e Recall

Se guardiamo le prestazioni in termini di recall dei diversi modelli notiamo che il livello più alto per i file vulnerabili è di 0,71 ottenuto da tre classificatori diversi, mentre per i file non vulnerabili il risultato migliore è stato ottenuto dal Random forest con un livello di 0,91, i risultati più bassi invece per i file vulnerabili sono stati registrati da SVM con 0,53, mentre per i file non vulnerabili il livello più basso è stato di 0,43 ottenuto da Naive Bayes. Attraverso la tabella 4.1 è anche possibile notare le differenze in termini prestazionali dei vari classificatori, in particolare notiamo che in generale il classificatore con risultati più

precisi è stato il Random forest seguito dal Decision tree, mentre il classificatore con risultati peggiori è stato il Naive Bayes, il quale ha registrato gli esiti più bassi per la predizione di file vulnerabili, registrando un grado di precisione molto basso.
Dalla figura 4.2 notiamo i risultati ottenuti in termini di F-measure dai diversi classificatori, questa misura attraverso un unico valore di permette di avere una visione più sintetica dei diversi risultati ottenuti dai classificatori, da questa misura notiamo che il classificatore che ha registrato maggiori prestazioni è stato il Random forest con un valore pari a 0,46, mentre i risultati peggiori sono stati ottenuti dal classificatore SVM con 0,22.

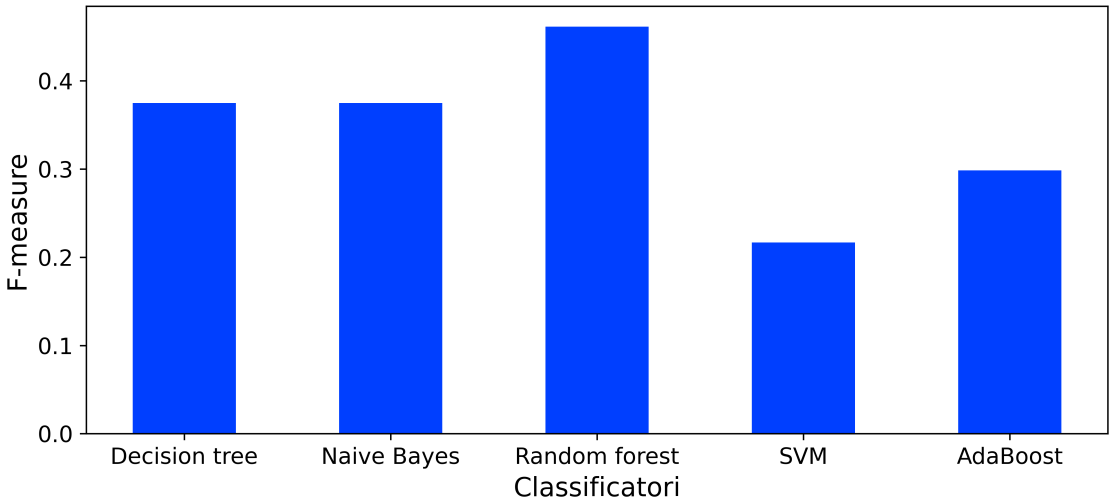

Figura 4.2: F-measure classificatori

Confrontando le figure 4.1 e 4.2 si possono notare risultati molto contrastanti tra F-measure e accuratezza, questa differenza è dovuta alla natura stessa del valore dell'accuratezza, quest'ultimo come descritto nel capitolo 3 mette in relazione tutte le predizioni corrette con tutte le predizioni fatte dal modello in fase di validazione, è possibile comprendere questi esiti notando che il problema affrontato è fortemente sbilanciato ed i vari classificatori si comportano molto bene nella predizione dei file non vulnerabili, quindi durante la fase di validazione gran parte dei file presi in considerazione per la predizione sono file non vulnerabili, tutto ciò implica che il numero totale di predizioni esatte aumenta incrementando anche il livello di accuratezza.

Di particolare importanza durante la fase di validazione di un modello di predizione è visualizzare e analizzare la matrice di confusione ottenuta dalla validazione del modello, quest'ultima ha lo scopo di mettere in relazione tutte le predizioni che il modello ha eseguito con tutti i risultati reali del test selezionato per la validazione, nelle seguenti figure 4.3, 4.4,

4.5, 4.6 e 4.7, sono stati riportati tutte le matrici di confusione di tutti i modelli implementati in questo studio. Leggendo questi dati possiamo notare che questi riflettono molto bene i dati analizzati precedentemente sopratutto valori di precision e recall analizzati nella tabella 4.1, inoltre ci permettono anche di comprendere il motivo di alcuni risultati ottenuti.

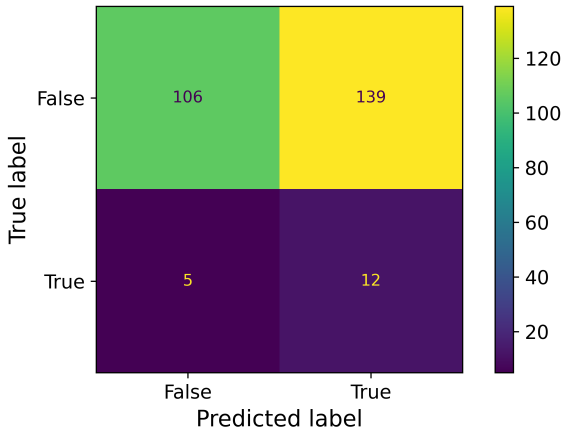

Figura 4.3: Matrice di confusione Naive Bayes

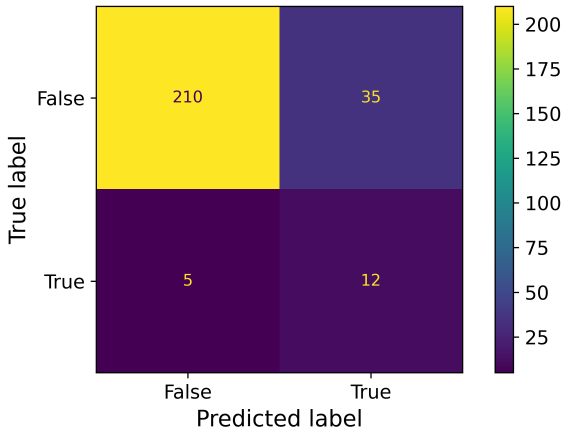

Figura 4.4: Matrice di confusione Decision Tree

Tra i dati più interessati emersi da questa analisi vediamo dalla matrice di confusione rappresentata dalla figura 4.3 notiamo che il seguente classificatore in questo contesto ha una forte carenza per la predizione di file non vulnerabili, questo forte deficit influenza notevolmente il risultato di precision ciò spiega il motivo per cui questo modello ha riscontrato i livelli di precision più bassi di tutti i modelli implementati, questa forte carenza spiega anche il motivo per cui il seguente modello ha riscontrato il peggiore risultato di recall per la predizione di file non vulnerabili. Nonostante tutto notiamo che il classificatore Naive Bayes per i file vulnerabili ha predetto correttamente lo stesso numero di file dei classificatori

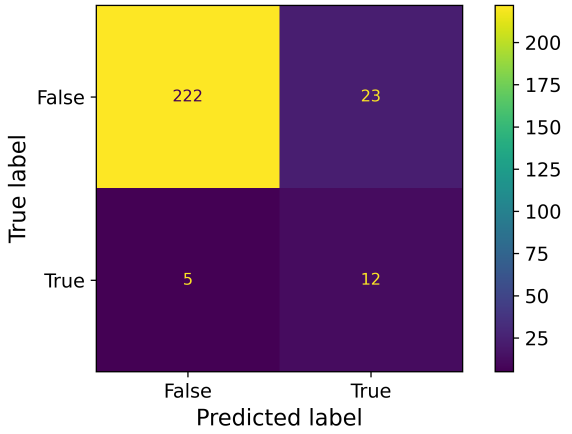

Figura 4.5: Matrice di confusione Random forest

che hanno raggiunto i risultati migliori e che sono Decision Tree 4.4 e Random Forest 4.5. Il modello di predizione che invece ha registrato una bassissima capacità di predizione per file vulnerabili da come si evince dalla figura 4.6 è stato il classificatore SVM con un numero di veri positivi migliore solo per una predizione nei confronti dei falsi positivi, nonostante ciò il classificatore ha ottenuto risultati migliori rispetto al classificatore Naive Bayes per la predizione dei file non vulnerabili, questo spiega il motivo per cui il SVM ha ottenuto risultati migliori in termini di precision. Come detto in precedenza, analizzando i diversi classificatori notiamo che i risultati migliori ottenuti per la predizione di file vulnerabili sono analoghi per diversi classificatori, quindi il risultato migliore in termini di precision è legato al classificatore che ha una capacità predittiva maggiore per i file non vulnerabili, infatti come si nota dalla figura 4.5 il classificatore Random forest ottiene i migliori risultati per la predizione di file non vulnerabili, ciò gli ha permesso di raggiungere i migliori risultati assoluti per il livello di precision.

Dalle seguenti analisi possiamo osservare come i diversi classificatori operano ottenendo in alcuni casi risultati molto diversi tra loro. Nel ambito della predizione come si evince da questo studio e confermato da studi precedenti analizzati nel capitolo 2 possiamo notare i classificatori più prestazionali per la predizione di vulnerabilità software sono quelli che prevedono la costruzione di uno o più alberi decisionali. Mentre il classificatore nel mio contesto che ha registrato performance più deludenti è stato il Naive Bayes, questo mostra come l'individuazione delle vulnerabilità software attraverso metriche software non è legato da fattori probabilistici.

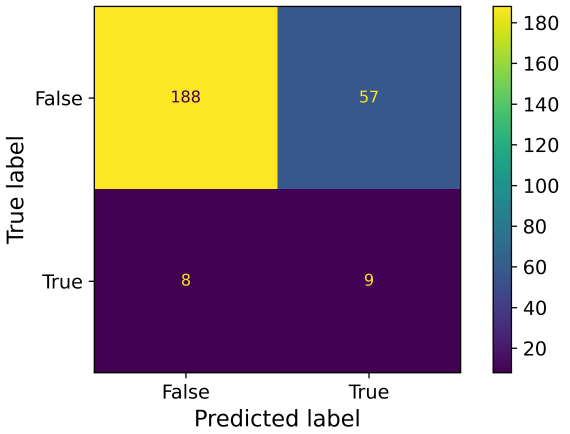

Figura 4.6: Matrice di confusione SVM

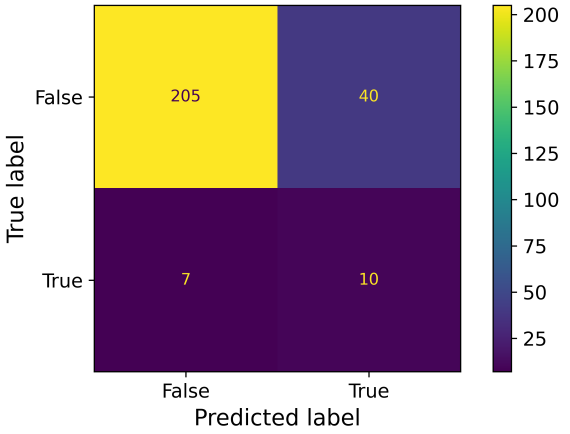

Figura 4.7: Matrice di confusione Ada Boost

CAPITOLO 5

Conclusioni e sviluppi futuri

In conclusione questo studio ha mostrato come e perché le vulnerabilità software coprono un ruolo di fondamentale importanza sul ciclo di vita del software e come queste possono influire notevolmente sulla vita degli utilizzatori, l'obiettivo principale di questo studio è stato di comprendere come le vulnerabilità possono essere individuate attraverso l'uso di VPM (modelli di predizione di vulnerabilità) analizzando come quest'ultimi lavorano e comprendendo il loro processo di implementazione. La parte più importante di questo lavoro è stata l'implementazione di un modello di predizione basato su machine learning che funzionasse sulla base di diverse metriche software, dalla costruzione del modello ho compreso come diverse metriche software possono lavorare insieme e quali di queste hanno un impatto maggiore, analizzando la loro correlazione con la presenza di vulnerabilità software. Dalla costruzione del modello ho potuto notare come i diversi classificatori lavorano e quali di questi hanno avuto un impatto migliore in termini di prestazioni per la predizione e il motivo legato questa differenza.

Questo lavoro apre molte porte di sviluppo, essendo come punto di partenza per altri studi, uno dei lavori futuri più importanti potrebbe essere l'ottimizzazione del modello in termini di correttezza delle predizioni, questa ottimizzazione potrebbe essere sviluppata attraverso l'uso diversi algoritmi di ottimizzazione e di ricerca. Un importante lavoro potrebbe essere l'implementazione di un prodotto software sul quale installare il modello di predizione e che renda l'utilizzo del modello più semplice e veloce, e fruibile su più dispositivi. Questo lavoro per varie vicissitudini ha compreso ed analizzato un numero ridotto di metriche

software anche se ne esistono un numero molto più grande ed alcune di queste raccolgono informazioni molto interessanti e che possono avere un impatto maggiore sulla corretta predizione del modello, per cui uno dei lavori più importanti è l'analisi di altre metriche software, con complessità diverse tra loro. Un importante sviluppo potrebbe essere la costruzione di un nuovo dataset, sulla base di quello usato da questo studio, che comprende diversi progetti con diverse complessità, integrando anche progetti software legati ad altri linguaggi di programmazione, in modo tale da creare un modello di predizione più preciso e complesso, il quale operi in modo corretto su più tipologie di prodotti software.

Ringraziamenti

Alla fine di questo percorso universitario voglio ringraziare la mia famiglia per il supporto e il sostegno che mi hanno mostrato e per tutti i sacrifici che hanno fatto per me, inoltre voglio ringraziare tutti i miei amici che mi hanno incoraggiato ed aiutato.
Voglio ringraziare il mio relatore Palomba Fabio per la sua disponibilità e per tutti i suoi preziosi consigli, inoltre voglio ringraziare tutti i membri del laboratorio SeSa lab, per la loro disponibilità e il loro aiuto.
Un ringraziamento speciale va a Dio che anche nei momenti più bui e difficili di questo percorso è sempre stato vicino a me, aiutandomi sempre ad andare avanti.
Voglio dedicare questo traguardo a Dio, alla mia famiglia e a tutti coloro che hanno sempre creduto in me.

Bibliografia

[1] P. d. C. dei Ministri, "Quadro strategico nazionale per la sicurezza dello spazio cibernetico," 2013. (Citato a pagina 1)

[2] R. Baldoni and R. De Nicola, "Il futuro della cyber security in italia," *Consorzio Interuniversitario Nazionale Informatica, November*, vol. 201, no. 5, 2015. (Citato a pagina 1)

[3] "Attacco hacker alla regione lazio: cosa sappiamo e cosa ci insegna." https://www.pandasecurity.com/it/mediacenter/sicurezza/attacco-regione-lazio/, 2021. (Citato a pagina 2)

[4] "Ransomware." https://www.garanteprivacy.it/temi/cybersecurity/ransomware. (Citato a pagina 2)

[5] "Sicurezza cibernetica." https://temi.camera.it/leg18/temi/sicurezza_cybernetica.html. (Citato a pagina 2)

[6] G. Italiano, "Piano nazionale di ripresa e resilienza (pnrr)," *Roma, Palazzo Chigi*, vol. 25, 2021. (Citato a pagina 2)

[7] "Direttiva nis." https://eur-lex.europa.eu/legal-content/IT/TXT/PDF/?uri=CELEX:32016L1148&from=NL. (Citato a pagina 3)

[8] "Direttiva nis2." https://www.europarl.europa.eu/RegData/etudes/BRIE/2021/689333/EPRS_BRI(2021)689333_EN.pdf. (Citato a pagina 3)

[9] "Clusit - associazione italiana per la sicurezza informatica." https://clusit.it/. (Citato a pagina 3)

[10] "Rapporto 2022 sulla sicurezza ict in italia - edizione di marzo 2022." https://clusit.it/rapporto-clusit/. (Citato a pagina 3)

[11] M. Bishop, "What is computer security?," *IEEE Security & Privacy*, vol. 1, no. 1, pp. 67–69, 2003. (Citato a pagina 5)

[12] "What is a vulnerability?." https://www.ictea.com/cs/knowledgebase.php?action=displayarticle&id=2092&language=italian. (Citato alle pagine 8 e 9)

[13] "Hardware vulnerability." https://www.cyber.gov.au/acsc/view-all-content/glossary/hardware-vulnerabilities. (Citato a pagina 9)

[14] "Vulnerability zero day." https://www.kaspersky.com/resource-center/definitions/zero-day-exploit. (Citato a pagina 9)

[15] "Remote code execution vulnerability in zoom client for windows." https://blog.0patch.com/2020/07/remote-code-execution-vulnerability-in.htmlhttps://blog.0patch.com/2020/07/remote-code-execution-vulnerability-in.html. (Citato a pagina 9)

[16] "Zoom confirms zero-day security vulnerability for windows 7 users." https://www.forbes.com/sites/daveywinder/2020/07/10/zoom-confirms-zero-day-security-vulnerability-for-windows-7-users/?sh=63abaaca753d. (Citato a pagina 9)

[17] "Owasp." https://owasp.org/. (Citato a pagina 9)

[18] "Buffer overflow." https://owasp.org/www-community/vulnerabilities/Buffer_Overflow. (Citato a pagina 9)

[19] "Owasp top 10." https://owasp.org/Top10/. (Citato a pagina 10)

[20] H. Saini, Y. S. Rao, and T. C. Panda, "Cyber-crimes and their impacts: A review," *International Journal of Engineering Research and Applications*, vol. 2, no. 2, pp. 202–209, 2012. (Citato a pagina 11)

[21] P. Shamal, K. Rahamathulla, and A. Akbar, "A study on software vulnerability prediction model," in *2017 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)*, pp. 703–706, IEEE, 2017. (Citato a pagina 11)

[22] J. Yang, D. Ryu, and J. Baik, "Improving vulnerability prediction accuracy with secure coding standard violation measures," in *2016 International Conference on Big Data and Smart Computing (BigComp)*, pp. 115–122, IEEE, 2016. (Citato a pagina 11)

[23] J. Walden, J. Stuckman, and R. Scandariato, "Predicting vulnerable components: Software metrics vs text mining," in *2014 IEEE 25th international symposium on software reliability engineering*, pp. 23–33, IEEE, 2014. (Citato a pagina 12)

[24] T. Zimmermann, N. Nagappan, and L. Williams, "Searching for a needle in a haystack: Predicting security vulnerabilities for windows vista," in *2010 Third international conference on software testing, verification and validation*, pp. 421–428, IEEE, 2010. (Citato a pagina 12)

[25] Y. Shin, A. Meneely, L. Williams, and J. A. Osborne, "Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities," *IEEE transactions on software engineering*, vol. 37, no. 6, pp. 772–787, 2010. (Citato a pagina 12)

[26] S. Neuhaus, T. Zimmermann, C. Holler, and A. Zeller, "Predicting vulnerable software components," in *Proceedings of the 14th ACM conference on Computer and communications security*, pp. 529–540, 2007. (Citato a pagina 12)

[27] V. H. Nguyen and L. M. S. Tran, "Predicting vulnerable software components with dependency graphs," in *Proceedings of the 6th International Workshop on Security Measurements and Metrics*, pp. 1–8, 2010. (Citato a pagina 12)

[28] R. Scandariato, J. Walden, A. Hovsepyan, and W. Joosen, "Predicting vulnerable software components via text mining," *IEEE Transactions on Software Engineering*, vol. 40, no. 10, pp. 993–1006, 2014. (Citato a pagina 12)

[29] C. Theisen and L. Williams, "Better together: Comparing vulnerability prediction models," *Information and Software Technology*, vol. 119, p. 106204, 2020. (Citato a pagina 13)

[30] F. Massacci and V. H. Nguyen, "Which is the right source for vulnerability studies? an empirical analysis on mozilla firefox," in *Proceedings of the 6th International Workshop on Security Measurements and Metrics*, pp. 1–8, 2010. (Citato a pagina 13)

[31] "CVE." `https://www.cve.org/`. (Citato alle pagine 13 e 16)

[32] "NVD." `https://nvd.nist.gov/`. (Citato a pagina 13)

[33] "MFSA." `https://www.mozilla.org/en-US/security/advisories/`. (Citato a pagina 14)

[34] "Redhat linux bugzilla." `https://bugzilla.redhat.com/`. (Citato a pagina 14)

[35] "Mozilla bugzilla." `https://bugzilla.mozilla.org/describecomponents.cgi`. (Citato a pagina 14)

[36] "Cloud foundry." `https://github.com/cloudfoundry/uaa`. (Citato a pagina 16)

[37] "Spring framework." `https://github.com/spring-projects/spring-framework`. (Citato a pagina 16)

[38] S. E. Ponta, H. Plate, A. Sabetta, M. Bezzi, and C. Dangremont, "A manually-curated dataset of fixes to vulnerabilities of open-source software," in *Proceedings of the 16th International Conference on Mining Software Repositories*, May 2019. (Citato alle pagine 16 e 18)

[39] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011. (Citato alle pagine 16 e 20)

[40] "Sesa lab." `https://sesalabunisa.github.io/en/index.html`. (Citato a pagina 18)

[41] S. Kaufman, S. Rosset, C. Perlich, and O. Stitelman, "Leakage in data mining: Formulation, detection, and avoidance," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 6, no. 4, pp. 1–21, 2012. (Citato a pagina 18)