UNIVERSITÀ DEGLI STUDI DI SALERNO

Dipartimento di Informatica

Corso di Laurea Triennale in Informatica

TESI DI LAUREA

Tecniche per individuare vulnerabilità software che mettono a rischio la privacy

RELATORE

Prof. **Fabio Palomba**

CORRELATORE

Dott. **Giammaria Giordano**

Università degli Studi di Salerno

CANDIDATO

Domenico D'Antuono

Matricola: 0512106538

Anno Accademico 2022-2023

"Il computer è la bicicletta della nostra mente"

Steve Jobs

Sommario

Una vulnerabilità è un debolezza del sistema informatico dovuta a un difetto del software, del protocollo di comunicazione o dell'hardware che può essere utilizzata per invalidare la sicurezza del sistema stesso. Alcuni tipi di vulnerabilità possono essere sfruttati per accedere illecitamente ai dati personali degli utenti e compromettere così la loro privacy, ad esempio quando il controllo degli accessi ad una applicazione viene interrotto. Nel tempo sono state proposte diverse tecniche per individuare vulnerabilità analizzando il codice sorgente anche tramite tecniche di machine learning, tali tecniche però non sono pensate per individuare le vulnerabilità sopra descritte e si limitano a distinguere tra codice sicuro e non sicuro, risultando per tale scopo poco efficaci. Questo lavoro di tesi si pone l'obiettivo di creare un modello di machine learning in grado di assegnare etichette specifiche a queste vulnerabilità aumentando così l'efficacia delle tecniche esistenti, per fare ciò verranno sfruttati i framework astminer e code2vec. I risultati dimostrano che è possibile assegnare etichette alle vulnerabilità di Controllo degli accessi interrotto migliorando l'efficacia degli strumenti attualmente disponibili, in particolare il modello ottiene prestazioni superiori al modello definito nella ricerca dx2021.

Indice

Elenco delle figure

Elenco delle tabelle

CAPITOLO 1

Introduzione

1.1 Contesto applicativo

La continua crescita di Internet e delle tecnologie che ne fanno uso hanno portato a grandi cambiamenti della nostra società con molte nuove opportunità per gli utenti e le aziende: possiamo parlare, condividere immagini, condividere video, condividere appunti, pubblicizzare prodotti, vendere prodotti. Questo indipendentemente dalla nostra posizione geografica.
Otre a portare questi vantaggi, internet è anche divenuta una nuova frontiera per le attività criminali.
Infatti, le attività criminali che si verificano online stanno continuamente evolvendosi, assumendo nuove e diverse forme, il cui impatto sociale è sempre più offensivo ed allarmante [2].
Questo ha portato anche a un cambio della legislazione in tal senso: con l'entrata in vigore del Trattato di Lisbona, la "criminalità informatica" è stata inclusa nell'articolo 83 del Trattato sul Funzionamento dell'Unione Europea (TFUE) come uno dei fenomeni criminali gravi e transnazionali di competenza penale dell'Unione Europea [2].
La privacy è una delle problematiche maggiori in questa nuova società bastata sul digitale, infatti, i nostri dati personali sono molto utili alle grandi aziende per scoprire le nostre abitudini e influenzare il nostro modo di pensare, votare, acquistare prodotti. Questi dati sono spesso facilmente accessibili ai big del settore e non solo.
Inoltre anche queste grandi aziende sono spesso vittime di attacchi, ad esempio, tramite i dati personali resi pubblici dagli utenti del noto social network Facebook qualche malintenzionato

	Nazione	Numero di utenti
1	Egitto	44.823.547
2	Tunisia	39.526.412
3	Italia	35.677.323
4	Usa	32.315.282
5	Arabia Saudita	28.804.686
6	Francia	19.848.559
7	Turchia	19.638.821
8	Marocco	18.939.198
9	Colombia	17.957.908
10	Iraq	17.116.398

Tabella 1.1: Utenti vittime dello scrap di Facebook

tramite un'attività di web scraping[1] è riuscito ad estrarre i dati di 533 milioni di utenti e renderli noti gratuitamente su un famoso sito di hacker, tra cui il 90% degli utenti italiani [3], come riportato nella Tabella 1.1.

Questi dati includevano nome, cognome, email, data di nascita, città, email e in alcuni casi i numeri di telefono degli utenti. Queste informazioni hanno avuto ampia diffusione e sono ora a disposizione dei criminali per tentare attività di truffa, tra le più diffuse c'è quella di sottrazione degli account.

Come le aziende private anche la pubblica amministrazione è continuamente impegnata ad operare con i dati dei cittadini e sono quindi spesso vittime di attacchi informatici: Nel 2021, la categoria "Gov" ha registrato il maggior numero di offese a livello globale, rappresentando il 15% del totale. Al secondo posto si trovano le Tecnologie dell'informazione e della comunicazione, con il 14%, seguite dalle categorie Assistenza sanitaria (13%), Educazione (9%) e Finanza/Assicurazioni (7%) [6].

Per contrastare tutte queste problematiche dopo anni di poca attenzione con il Decreto-legge del 14 giugno 2021 il Governo ha tra l'altro istituito l'ACN, l'Agenzia per la Cybersicurezza Nazionale a tutela degli interessi nazionali nel cyberspazio [7].

[1] Il web data scraping è una particolare tecnica per l'estrazione di dati dai siti web mediante strumenti automatizzati.

Inoltre, questi attacchi stanno aumentando il loro livello di pericolosità, come si può vedere dal grafico della Figura 1.1, nel 2021 i cyber attacchi di livello Critico hanno costituito il 32% del totale, mentre quelli di livello Alto sono stati riscontrati nel 47% dei casi. Gli attacchi con impatto Medio hanno invece rappresentato il 19% del totale. Di conseguenza, nel corso dell'anno scorso, gli attacchi di livello Critico e Alto hanno costituito quasi l'80% di tutti gli attacchi informatici [8]. Da tenere in considerazione che un attacco di livello critico può mettere addirittura a serio rischio la sopravvivenza di un organizzazione di grandi dimensioni.

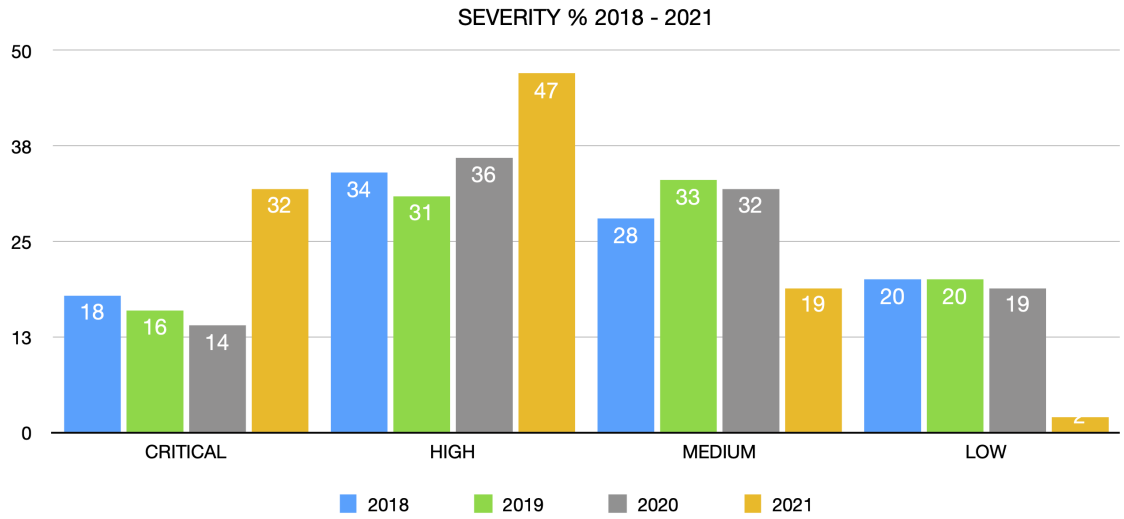

Figura 1.1: Percentuali Severity negli anni

Per quanto riguarda invece le tecniche di attacco, come mostrato nella Figura 1.2, nel 2021 la categoria "Malware" è rimasta la più frequente, rappresentando il 41% di tutti gli attacchi. Al secondo posto, con il 21%, si trova la categoria "Unknown", che si riferisce principalmente a eventi di Data Breach[2] . Al terzo posto, con il 16% del totale, si trovano i cyber attacchi di tipo "Vulnerabilities", in netta crescita. La categoria "Phishing/Social Engineering" occupa il quarto posto, rappresentando il 10% del totale. Tutte le altre tipologie di attacco, tra cui "Multiple Techniques" e "Account Cracking", insieme rappresentano il 22% del totale degli attacchi cyber registrati nel 2021 [8].

Una tecnica di attacco rientra nella categoria "Malware" quando questa si basa sull'installazione di un software malevolo che può a sua volta rendere il sistema vulnerabile ad altri attacchi, in "Vulnerabilites" rientrano invece tutte le tecniche che sfruttano vulnerabilità note.

[2]Un evento di data breach è una violazione di sicurezza che comporta, accidentalmente o in modo illecito, la distruzione, la perdita, la modifica, la divulgazione non autorizzata o l'accesso ai dati personali trasmessi, conservati o trattati.

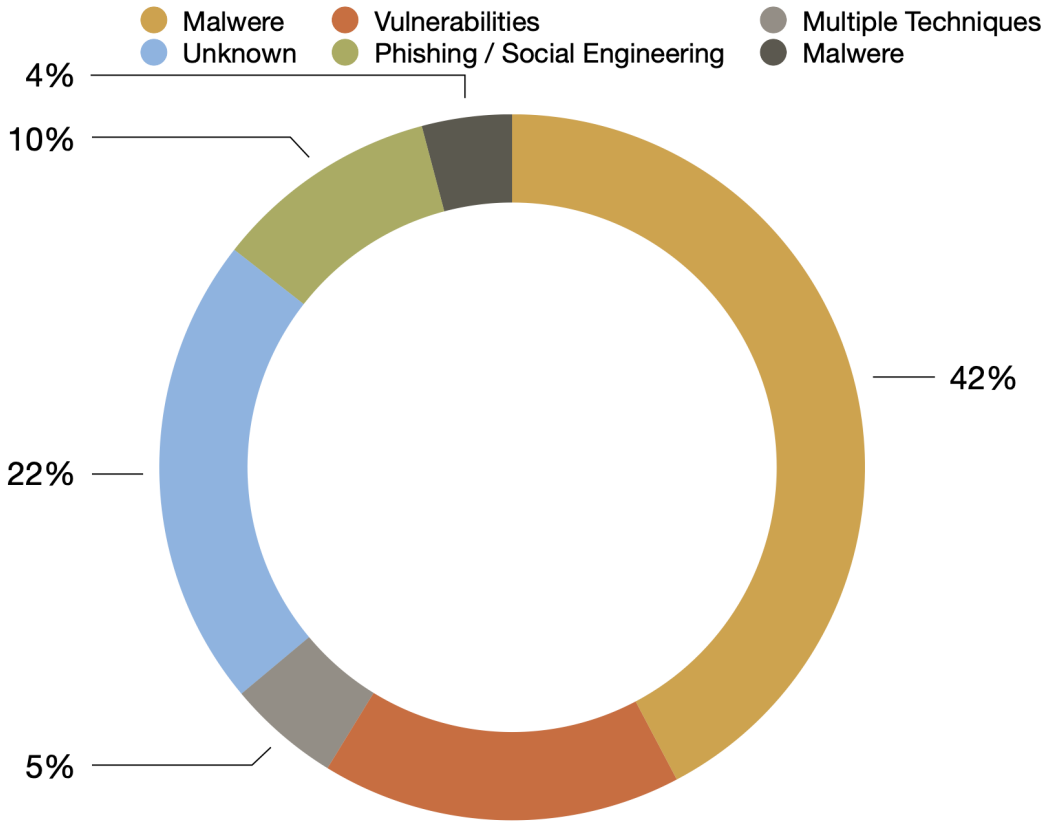

Figura 1.2: Percentuali tecniche usate per attaccare la privacy

Quindi come nel caso trattato sopra di Facebook, vengono spesso utilizzati errori che sono stati commessi quando si sono definiti i requisiti del sistema software che risultano quindi ambigui su alcuni aspetti, queste ambiguità possono indurre vulnerabilità che portano a problemi critici per il sistema software e gravi rischi per la privacy.

1.2 Motivazioni e Obiettivi

Come si può vedere dalla sezione introduttiva i problemi la sicurezza informatica è diventata cruciale per prevenire gravi danni alla vita di tutti, la violazione della privacy può comportare seri rischi per infrastrutture e aziende e la vita privata di persone comuni. Per sottrarre questi dati vengono spesso sfruttate delle vulnerabilità software, il mio lavoro andrà ad analizzare le tecniche esistenti per individuare vulnerabilità (generiche e specifiche per

la privacy) e le loro evoluzioni: tecniche di machine learning. Per poi proporre una tecnica basata su machine learning, specializzata nell'identificazione di un tipo di vulnerabilità che metta a rischio la privacy. L'obiettivo è quello di mettere a disposizione uno strumento che potrà essere integrato a quelli esistenti per prevenire il più possibile l'introduzione di problemi di privacy, il modello di machine learning ottenuto, dato in input un codice sorgente in linguaggio C, sarà in grado di individuare le sezioni di codice vulnerabili ad azioni fraudolente per prendere il controllo di aree personali riservate. Un tale modello potrebbe ottenere prestazioni superiori ai modelli già esistenti che sono stati addestrati per distinguere solo codice vulnerabile e non vulnerabile. Parti del codice sorgente che contengono queste vulnerabilità presentano caratteristiche comuni che il modello potrà imparare a riconoscere una volta addestrato. Per fare quanto descritto il codice sorgente deve essere prima trasformato in AST (che vedremo in dettaglio nel capitolo 2.1.2), gli AST avranno un etichetta che verrà utilizzata da code2vec, in versione opportunamente modificata, per trovare similitudini e addestrare un modello in grado di riconoscerle. La versione originale del tool è utilizzata per individuare percorsi in un codice sorgente e assegnare etichette comuni alle istruzioni che fanno parte dello stesso percorso logico, la versione qui proposta invece utilizzerà questa capacità di individuare similitudini per riconoscere le vulnerabilità di Controllo degli accessi interrotto.

1.3 Risultati

I risultati dello studio rivelano che è possibile utilizzare la combinazione dei framework Astminer[9] e code2vec[10] per identificare vulnerabilità che comportano un rischio per la privacy in particolare quelle riguardanti il Controllo degli accessi interrotto, ispirandomi alla ricerca dx2021[1] che utilizza questi framework per identificare vulnerabilià generiche. Infatti, è possibile osservare come l'assegnazione di una nuova etichetta dedicata alle istanze che presentano un problema di privacy mostra risultati superiori a quelli ottenuti tramite il modello sviluppato in dx2021[1] che è in grado di identificare solo generiche vulnerabilità. É importante evidenziare che questo risultato è stato ottenuto ottimizzando gli iperparametri del framework code2vec[10]. I risultati migliori sono stati ottenuti dal modello che prevedeva un raddoppio del parametro di TRAIN_BATCH_SIZE.
Questo lavoro dovrebbe essere ampliato, un possibile ampliamento potrebbe essere apportato analizzando i risultati ottenuti tramite un dataset ancora più ampio per il Controllo degli accessi interrotto trattato in questa ricerca. Inoltre, negli studi successivi potrebbero essere

prevista l'identificazione di altri tipi di vulnerabilità che determinano un rischio per la privacy (tutte quelle indicate in [11]) e sarebbe interessante provare a mettere insieme le varie ricerche di classificazione delle vulnerabilità (incluso questa e [12]) e definire un modello in grado di prevedere tutte le etichette legate ai vari tipi di vulnerabilità della privacy, per analizzare poi le prestazioni così ottenute dal framework.
In sintesi, il lavoro di tesi ha fornito i seguenti contributi:

- Da CVEfixes[24] è stato ricavato il dataset fixes che classifica le vulnerabilità di Controllo degli accessi interrotto come legate alla privacy.
- Studio e analisi di una tecnica di identificazione di vulnerabilità legate alla privacy, in particolare quelle che riguardano il Controllo degli accessi interrotto, basata sull'utilizzo dei framework astminer[9] e code2vec[10];
- Sono emersi diversi spunti interessanti sull'ottimizzazione degli iperparametri di code2vec[10], è stato così ottenuto il modello fixmodel0.3 che ottiene risultati importanti sul dataset fixes.

1.4 Struttura della tesi

La Sezione 2 illustra la letteratura correlata all'identificazione delle vulnerabilità sia generiche che quelle strettamente legate alla privacy e approfondisce degli aspetti relativi ai concetti trattati nella tesi. La Sezione 3 riporta la metodologia e le tecniche utilizzate per raggiungere gli obiettivi della tesi, mentre la Sezione 4 riporta i risultati ottenuti dalle varie esecuzioni con variazioni agli iperparametri e dal modello di confronto. La sezione 5 conclude la tesi con l'analisi dei risultati e alcune riflessioni. I file della ricerca sono stati caricati su GitHub al seguente link: https://github.com/powerdom00/dantuono_privacy_detection.

CAPITOLO 2

Background e Stato dell'Arte

2.1 Background

2.1.1 Definizioni di vulnerabilità

In questo capitolo vedremo le vulnerabilità più utilizzate per mettere a rischio la privacy.

- Controllo degli accessi interrotto[11]: I controlli di accesso sono fondamentali per proteggere le applicazioni dall'accesso non autorizzato a dati e risorse. Controlli di accesso interrotti possono portare alla compromissione dei dati, all'ottenimento di autorizzazioni oltre a quelle previste per gli utenti standard o ad attacchi di acquisizione di account in cui estranei dirottano gli account utente e avviano transazioni fraudolente. Questa vulnerabilità è passata dalla quinta posizione nel 2017 alla prima nel 2021, riflettendo che è stata rilevata nel 94% delle applicazioni testate. Le vulnerabilità comuni in questa categoria di rischio includono errori della logica dell'applicazione che ignorano i controlli di controllo degli accessi consentendo agli utenti di modificare i valori dei parametri o forzare la navigazione verso determinati URL.

- Errori crittografici[11]: Si riferiscono a una scorretta implementazione della crittografia o a una completa mancanza di crittografia. La principale conseguenza di un errore crittografico è che puoi potenzialmente esporre dati sensibili. Una cattiva implementazione può essere, ad esempio, quella nella quale vengono utilizzati parametri crittografici vulnerabili come una chiave corta. I processori sempre più veloci che abbiamo a dispo-

sizione possono attaccare queste implementazioni con la cosiddetta tecnica di forza bruta: si generano tutte le possibili chiavi.

- Iniezione[11]: È una categoria di rischio che si riferisce alla capacità degli attori delle minacce di fornire input dannosi alle applicazioni Web che fanno sì che l'app esegua comandi imprevisti e indesiderati. L'iniezione si verifica quando l'app non riesce a distinguere l'input dannoso dal suo codice. Gli attacchi di iniezione comuni includono iniezioni SQL che inseriscono query SQL dannose nei campi di input o iniezioni JavaScript che caricano codice dannoso nel lato client dell'app Web. Gli attacchi di iniezione possono portare a vari esiti negativi, tra cui denial of service, elevazione dei privilegi e violazioni dei dati.
  Con denial of service ci si riferisce a un malfunzionamento dovuto a un attacco informatico in cui si fanno esaurire deliberatamente le risorse di un sistema informatico che fornisce un servizio ai client.
  Con elevazione dei privilegi si intende invece, per esempio, la possibilità che un malintenzionato possa modificare il tipo di account dal quale interagisce con il sistema, un tipo di account diverso può accedere ad aree riservate dell'applicazione che possono contenere dati riservati. Di seguito viene mostrato un esempio di iniezione SQL, in cui un utente malintenzionato invia OR 1=1– invece di un ID valido, come mostrato nella Figura 2.1. Senza una convalida dell'input, questa query restituirà tutti i dati contenuti nella tabella, in questo caso il conti da tavolo [12].

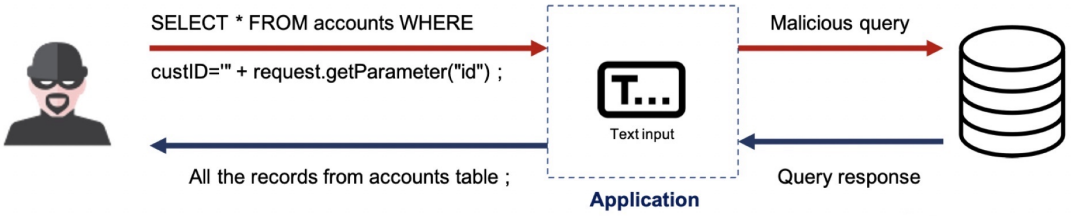

Figura 2.1: Esempio di iniezione

Come si può capire dall'esempio, tramite l'iniezione, i malintenzionati hanno la possibilità di accedere nell'area utente dell'applicazione, di conseguenza la privacy risulta del tutto compromessa.

- Design insicuro[11]: Quando un'applicazione è intrinsecamente progettata in modo non sicuro, anche una perfetta implementazione dei controlli di sicurezza e dei rischi non può compensare tali debolezze di progettazione. I sofisticati attori delle minacce alla fine troveranno e sfrutteranno i difetti di progettazione.

- Errata configurazione della sicurezza[11]: Questa categoria di rischi si riferisce ai componenti di sicurezza in un'applicazione configurati in modo errato. Le configurazioni errate sono sempre più comuni a causa dell'utilizzo del cloud come ambiente di sviluppo e della creazione di app Web con immagini di container. La complessità infrastrutturale aggiunge più punti in cui possono verificarsi errori di configurazione della sicurezza. Qui sono spesso in gioco singoli errori, come aprire porte non necessarie, non modificare le password predefinite o lasciare aperti i bucket di archiviazione cloud. Un cambiamento strategico fondamentale consiste nel garantire un processo ripetibile per il rafforzamento delle configurazioni e uno strumento o processo che controlli e verifichi automaticamente tali configurazioni negli ambienti on-premise e cloud.

- Componenti vulnerabili e obsoleti[11]: Le app Web comprendono molti componenti o blocchi costitutivi da fonti esterne (librerie, framework, ecc.). Questi componenti gestiscono sia la funzionalità di back-end che quella di front-end. Quando gli attori delle minacce tentano di compromettere un'applicazione, esaminano le sue parti componenti e tentano di sfruttare eventuali vulnerabilità. Spesso, queste vulnerabilità derivano dall'utilizzo di framework o librerie non aggiornati che sono facili da sfruttare.

- Errori di identificazione e autenticazione[11]: Gli errori nell'autenticazione e nella gestione dell'identità rendono le applicazioni vulnerabili agli autori delle minacce che si spacciano per utenti legittimi. Alcuni esempi di vulnerabilità includono non impostare periodi di validità per gli ID di sessione, consentire password deboli facili da indovinare e non limitare i tentativi di accesso contro attacchi automatici. Le soluzioni includono l'implementazione dell'autenticazione a più fattori nelle app e la comunicazione agli sviluppatori dell'importanza di rispettare la lunghezza, la complessità e le politiche di rotazione delle password consigliate.

- Errori di integrità del software e dei dati[11]: Si tratta di formulare ipotesi predefinite errate all'interno delle pipeline di sviluppo sull'integrità del software o dei dati. Poiché le app Web si basano regolarmente su plug-in e librerie provenienti da fonti esterne, la mancanza di verifica dell'integrità di queste fonti introduce il rischio di codice dannoso,

accesso non autorizzato e compromissione. La principale strategia di mitigazione è garantire che il codice esterno o i dati non siano stati manomessi richiedendo firme digitali.

- Registrazione della sicurezza e monitoraggio degli errori[11]: La registrazione e il monitoraggio aiutano a fornire responsabilità sulla sicurezza, visibilità sugli eventi, avvisi di incidenti e analisi forensi. Quando si verificano errori in queste funzionalità, la capacità delle aziende di rilevare e rispondere alle violazioni delle applicazioni viene gravemente compromessa. Per mitigare, utilizzare strumenti open source o proprietari per correlare i registri, implementare il monitoraggio e gli avvisi e creare una strategia di ripristino e risposta agli incidenti utilizzando linee guida stabilite, come NIST 800-61r2.

- Falsificazione delle richieste lato server (SSRF)[11]: La maggior parte delle app Web oggi richiede risorse esterne per la loro funzionalità, a cui di solito si accede tramite URL. SSRF si verifica quando gli hacker possono convincere i server a fare richieste che controllano. La vulnerabilità tipica è che l'applicazione Web non convalida l'URL fornito dall'utente, consentendo potenzialmente l'accesso a servizi o risorse interni aggirando i controlli di accesso. Il concetto strategico di difesa in profondità è importante qui; più controlli a livello di applicazione e rete possono aiutare a prevenire SSRF. I dati di input forniti dall'utente devono essere convalidati e disinfettati, mentre anche la segmentazione della rete può essere d'aiuto.

  Un esempio di vulnerabilità legata alla crittografia è il bug Heartbleed che ha interessato la libreria di software crittografico OpenSSL. Questa debolezza consente di rubare delle informazioni protette dalla crittografia SSL/TLS. L'SSL/TLS fornisce sicurezza e privacy nelle comunicazioni su Internet per applicazioni quali Web, posta elettronica, messaggistica istantanea (IM) e alcune reti private virtuali (VPN). Dopo la scoperta, Google ha istituito Project Zero, un organizzazione che ha il compito di trovare vulnerabilità zero-day per proteggere il web. Sono state rilasciate su Linux patch per proteggere la vulnerabilità OpenSSL [13].
  Nel momento della rivelazione, qualcosa come il 17% (circa mezzo milione) dei web server sicuri in internet, certificati dalle Certification Authority diventarono vulnerabili agli attacchi, permettendo il furto delle chiavi private dei server e i cookie di sessioni e le password degli utenti. Electronic Frontier Foundation, Ars Technica e Bruce Schneier ritennero il bug Heartbleed catastrofico [14].

2.1.2 Tecniche di rappresentazione del codice sorgente

Per rendere il codice sorgente comprensibile a un modello di machine learning, il codice va opportunamente trasformato, una possibilità consiste nell'utilizzare il concetto di AST per ottenere i relativi AST path e path-context.
AST [1]: Un albero della sintassi astratta (AST) per un frammento di codice C è una tupla < N, T, X, s, δ, ϕ> dove N è un insieme di nodi non terminali, T è un insieme di nodi terminali, X è un insieme di valori, $s \in N$ è il nodo radice, $\delta : N \rightarrow (N \cup T)$è una funzione che mappa un nodo non terminale a una lista dei suoi figli, $\phi : T \rightarrow X$ è una funzione che mappa un nodo terminale al suo valore associato. Ogni nodo tranne la radice appare esattamente una volta in tutte le liste dei figli; cioè, ogni nodo ha esattamente un genitore.
AST path [1]: Un cammino AST di lunghezza k è una successione della forma: n1d1 . . . nkdknk+1, dove $n1, nk + 1 \in T$ sono nodi terminali, $\forall i \in [2..k] : ni \in N$ sono nodi non terminali e $\forall i \in [1..k] : di \in \{\uparrow, \downarrow\}$ sono le direzioni di movimento (su o giù nell'albero). Se $di = \uparrow$, allora $ni \in \delta(ni + 1)$; se $di = \downarrow$, allora $ni + 1 \in \delta(ni)$.
Path-context [1]: Dato un cammino AST p, un percorso-contesto è una tripletta
< xs , p, xt > dove p è un cammino sintattico nell'AST e xs e xt corrispondono rispettivamente ai valori associati con i terminali iniziale e finale di p.
Un possibile path-context per l'istruzione x = 7 sarebbe:
$< x, (EspressioneNome \uparrow EspressioneAssegna \downarrow EspressioneIntegerLetterale), 7 >$

2.2 Stato dell'arte

2.2.1 Tecniche consolidate per individuare vulnerabilità software

Le vulnerabilità software come visto nei primi capitoli sono una delle principali cause di attacchi informatici per rubare i dati e compromettere la privacy degli utenti. Per determinare quali sono le vulnerabilità di un determinato software esistono diverse tecniche, più o meno recenti e avanzate. Le tecniche elencate di seguito sono state pensate per individuare problemi generici tra i quali può rientrare il furto dei dati e quindi la violazione della privacy anche se non sono state pensate nello specifico per individuare tali vulnerabilità.

Analisi statica: È il processo di valutazione di un sistema o un componente in base alla sua forma, struttura, contenuto o documentazione, che non richiede l'esecuzione del programma [15]. Questo tipo di analisi costituisce un problema indecidibile come afferma il teorema di Rice, di conseguenza darà luogo a falsi positivi e falsi negativi e non possiamo ottenere un risultato assoluto ma entro questi limiti resta comunque molto utile per il suo scopo. Per realizzare tali analisi alcuni linguaggi di programmazione mettono a disposizione tool appositi.
I correttori di codice statico in Java sono disponibili in due versioni: quelli che funzionano direttamente sul codice sorgente del programma e quelli che funzionano sul bytecode compilato. Sebbene ogni controllo del codice funzioni a modo suo, la maggior parte condivide alcune caratteristiche di base. Leggono il programma e ne costruiscono un modello, una specie di rappresentazione astratta che possono utilizzare per abbinare i modelli di errore che riconoscono. Eseguono anche una sorta di analisi del flusso di dati, cercando di dedurre i possibili valori che le variabili potrebbero avere in determinati punti del programma [16].

Fuzzing: Miller et al. ha proposto per primo questo tipo di approccio, utilizzando questa tecnica su 90 programmi UNIX è stato in grado di bloccarne il 24%.
Consiste nel dare un input non previsto a un programma, in particolare si generano dati semi-validi o casuali, questi dati vengono inviati alla applicazione di destinazione, infine si osserva il comportamento dell'applicazione per vedere se fallisce mentre sta consumando i dati [15]. I dati semi-validi sono dati sufficientemente corretti da superare gli esami di input ma allo stesso tempo possono causare problemi al programma testato. Per generare i dati dopo una prima fase dove si utilizzavano dati randomizzati che venivano facilmente respinti dai controlli sugli input si è passati alla generazione e alla mutazione. La tecnica di generazione dei dati si basa solitamente su specifiche, come le specifiche del formato di file, ad esempio Autodafe e APIKE Proxy, mentre la tecnica di mutazione va a modificare alcuni campi di input accettati dal programma (la prima richiede una conoscenza delle specifiche di input approfondita) [15]. Per monitorare i risultati vengono di solito utilizzati tool di debug come OllyDbg e GDB.

Test di penetrazione: Valuta la sicurezza di un sistema simulando gli attacchi di utenti malintenzionati e valutando se gli attacchi hanno esito positivo.
Un test di penetrazione richiede un'analisi dettagliata delle minacce e dei potenziali ag-

gressori per essere più prezioso, l'utilizzo dei risultati dei test di penetrazione richiede un'interpretazione adeguata [17].

Tutti i tipi di test possono mostrare solo la presenza di difetti e mai l'assenza di essi, Il meglio che i tester possono dire è che i difetti specifici che hanno cercato e che non sono riusciti a trovare non sono presenti: questo può dare un'idea della sicurezza generale della progettazione e dell'implementazione del sistema [17].

Il test di penetrazione viene effettuato prima della pubblicazione e può essere di 2 tipi:

Black box: si tratta di un tentativo di attacco da chi non conosce il sistema

White box: chi attacca conosce il codice sorgente o l'architettura interna del sistema software, un attacco del genere può avvenire per via di furto di dati aziendali.

Il piano di test dovrebbe basarsi sull'analisi del rischio e sulla modellazione delle minacce [15], per modellare le minacce ci si può avvalere di STRIDE "("[18]metodologia di modellazione delle minacce per garantire la sicurezza delle applicazioni"), presentando i risultati come Threat Trees.

Dopo ciò si vanno a prevedere i test di penetrazione, per quanto riguarda i tester che eseguono il test, si raccomanda il PTES (Penetration Testing Execution Standard).

PTES consiste principalmente di sette fasi: a. Interazioni di pre-impegno; b. Raccolta di informazioni; c. Modellazione delle minacce; d. Analisi delle vulnerabilità; e. Sfruttamento; f. dopo lo sfruttamento; g. Segnalazione [15].

Nella fase di segnalazione è bene riportare informazioni come: fasi di riproduzione, gravità, scenari di exploit ed esempi di codice di exploit; possono risultare fondamentali per valutare il livello di rischio per la sicurezza e creare un nuovo piano di pubblicazione del prodotto.

Infine possiamo dire che per effettuare un penetration testing di qualità è buona pratica presupporre che chi attacca conosce il codice sorgente e quindi andare verso un testing White box come suggerito dalle best practice OWASP (insieme di tecniche per garantire la qualità del codice [19]), OSSTMM(manuale open source per l'esecuzione dei test di sicurezza verso infrastrutture ed asset informatici [20]).

Nonostante tutto bisogna tenere in considerazione che anche un buon esito di questi test non può mai stabilire una invulnerabilità del sistema (il problema è indecidibile), Il continuo utilizzo delle vulnerabilità del software per attaccare sistemi e privacy rendono questi strumenti utili ma non sufficienti e vanno esplorate le nuove possibilità offerte dal machine learning per aumentare la sicurezza senza intaccare la sostenibilità economica dei progetti software.

2.2.2 Machine learning per il rilevamento di vulnerabilità

Il machine learning è sempre più utilizzato in qualsiasi ambito, vediamo i modi in cui è possibile utilizzarlo per il rilevamento e la correzione di vulnerabilità. Nelle tecniche che utilizzano machine learning per rilevare vulnerabilità il rilevamento è di solito eseguito come classificazione mentre la correzione come generazione.
Il rilevamento automatico è un processo di creazione di classificatori per individuare le parti del codice che contengono potenziali difetti basandosi sulla complessità del codice e sulla cronologia delle modifiche [21], per la correzione automatizzata si deve invece identificare automaticamente una patch da applicare al problema individuato con scarso intervento umano. Quest'ultima è una cosa molto più difficile da ottenere e di solito si basa sull'idea di partire da un esempio corretto per trovare le differenze e cambiare il codice in base a questo, gli errori da correggere come per i tool tradizionali sono sintattici e semantici.
La creazione di un modello di machine learning per questo scopo prevede solitamente raccolta e preparazione dei dati, formazione del modello e, infine, valutazione e distribuzione [21]. Nella prima fase si procede a trasformare il codice sorgente in una rappresentazione di basso livello per il modello di machine Learning ad esempio vettori per reti neurali, ecco le modalità più comuni;
Sequenza di token: suddivisione del codice in unità più piccole.
Albero della sintassi astratta: i costrutti del programma diventano nodi dell'albero.
Grafici: possono acquisire varie relazioni sintattiche e semantiche e proprietà del codice sorgente.
Il modello va addestrato con un set di dati di qualità, ad esempio un seti di programmi metà con bug e metà senza potrebbe produrre errori su programmi reali.
Un problema comune che emerge durante la valutazione e la replica dei risultati è l'overfitting: il modello si adatta troppo ai set di apprendimento [21], perciò la selezione delle caratteristiche rilevanti è uno dei compiti più importanti di machine learning e inserire troppo funzioni non è necessariamente meglio.
La previsione di un modello machine learning ha quattro possibili stati di classificazione, ovvero la matrice di confusione: true positive (TP), true negative (TN), false positive (FP) e false negative (FN) [21].
Le tecniche sviluppate fino ad ora utilizzano come rappresentazione alberi e token per rappresentare il codice, si dividono quasi equamente tra analisi sintattica e semantica e

evidenziano buoni risultati ma è sempre bene tenere a mente che i set di addestramento devono essere aderenti il più possibile a scenari reali.

Una ricerca interessante di questo tipo è [22] dove è stato messo a punto uno strumenti di rivelazione di vulnerabilità del codice sorgente in C e C++. In questo caso il set di dati è stato creato tramite un analisi statica del codice sorgente preso dai repository Debian e GitHub, il set è stato poi esteso con il dataset SATE IV. A questi dati sono stati applicati algoritmi di machine learning sia CNN (caratteristiche convoluzionali) che RNN (funzionalità ricorrenti) ma i risultati migliori sono stati ottenuti con il primo approccio: Nel complesso, i nostri modelli CNN hanno ottenuto risultati migliori rispetto ai modelli RNN sia come classificatori autonomi che come generatori di funzionalità [22]. L'articolo [23] va invece nella direzione dell'analisi del codice a livello di astrazione più basso e tramite un'analisi dinamica del codice binario: sono state raccolte 9872 sequenze di chiamate di funzione come caratteristiche per rappresentare i modelli dei programmi binari durante la loro esecuzione [23], le reti neurali utilizzate sono la già citata CNN, la LSTM e un ibrido tra le due. I risultati sulla rilevazione di vulnerabilità hanno portato ad un'accuratezza dell' 83,6%.

Le tecniche prese in considerazione fino a questo punto sono utilizzate per individuare vulnerabilità generiche e non sono quindi specifiche per la privacy, in questo senso esistono poche ricerche mirate oltre che all'identificazione di vulnerabilità anche alla loro classificazione.

Uno studio in questo senso che sono riuscito a trovare è [12], in questo caso il modello di machine learning oltre che a individuare potenziali vulnerabilità nel codice sorgente va a classificarle, un esempio che viene riportato è quello dell'injection.

Il lavoro in questione partendo da uno studio sull'identificazione di vulnerabilità basato sul tool open source code2seq ha previsto delle modifiche per fare in modo che il modello di apprendimento fosse indirizzato verso uno specifico tipo di vulnerabilità. In particolare nella prima fase, poiché l'obiettivo del nuovo Fastscan era la creazione di un modello in grado di rilevare una vulnerabilità specifica, è stato necessario filtrare i dati di input del set di dati originale per tipo di vulnerabilità. In modo che sia possibile addestrare un modello per prevedere solo un tipo specifico di vulnerabilità [12]. Quanto descritto è stato realizzato con linguaggio python leggendo i file xml dei vari progetti e classificandoli.

Nell'articola si evidenzia l'importanza dell'ottimizzazione degli iperparametri, questo può portare a miglioramenti nelle metriche di valutazione dei risultati, tra le quali troviamo:

La metrica Precision: Si riferisce al rapporto tra le osservazioni positive previste correttamente e le osservazioni positive totali previste. Questa metrica risponde alla domanda dei metodi etichettati come aventi una vulnerabilità, quanti di loro ne avevano effettivamente uno? Un'elevata precisione significa una bassa occorrenza di falsi positivi [12];

La metrica Recall: È spesso chiamato sensibilità, risponde alla domanda, dai metodi che avevano davvero una vulnerabilità, quanti di loro sono stati etichettati? [12];

La metrica F1: È la media ponderata tra precisione e richiamo, questa è la misura più bilanciata di tutte le preimpostate poiché tiene conto dei falsi positivi e dei falsi negativi. Questa metrica è utile quando c'è una distribuzione non uniforme nel set di dati, nel caso in cui non ci sia un numero ravvicinato tra voci vulnerabili e non vulnerabili [12].

Per quanto riguarda i risultati ottenuti: Il modello finale per l'iniezione che utilizzava il set di dati dt01 aveva l'85% di Accuracy, il 90% di Precision, il 97% di Recall che portava a un F1 del 93% [12].

CAPITOLO 3

Una Tecnica di Identificazione di Vulnerabilità Relative alla Privacy Basata sull'utilizzo dei tool Astminer e Code2Vec

Il presente capitolo tratterà un'approccio basato sulla classificazione delle vulnerabilità generiche e sull'utilizzo di Astminer[9] e Code2vec[10] per l'identificazione di vulnerabilità software che rappresentano rischi per la privacy.

3.1 Approccio

Come descritto nello stato dell'arte, esistono pochi lavori sulla classificazione delle vulnerabilità. Uno di questi è [12], in questo caso si è deciso di identificare le vulnerabilità relative al cosiddetto injection. Come visto nella sezione "definizioni di vulnerabilità", ne esistono molti altri tipi che possono impattare sulla privacy. Tra queste risulta al primo posto del rapporto Clusit il "[11] Controllo degli accessi interrotto", questo è dovuto a errori nella logica dell'applicazione che consentono ai malintenzionati di ottenere accesso ad account di altri utenti. Ho deciso quindi di classificare le istanze del dataset indicando "Privacy" come target quando si tratta di una funzione che rientra nella categoria Controllo degli accessi interrotto.
Partendo dal database messo a disposizione da CVEfixes Dataset[24] che contiene metodi c, ho ottenuto un dataset quasi compatibile con Astminer[9] nella versione modificata nella ricerca dx2021[1], quindi ho modificato Astminer[9] per assegnare

ai path-context estratti il nuovo tipo di etichetta dedicata alla Privacy. Infine come in dx2021[1] ho utilizzato i path-contexts estratti e etichettati per addestrare un nuovo modello di machine learning tramite il framework Code2Vec[10]. La Figura3.1 sintetizza il processo utilizzato, si parte da un dataset composto da funzioni vulnerabili, non vulnerabili e privacy vulnerabili, si usa astminer per estrarre gli AST dai quali vengono ricavati i context-path (questo per tutti i tipi di target), questi vengono poi passati a code2vec per l'addestramento.

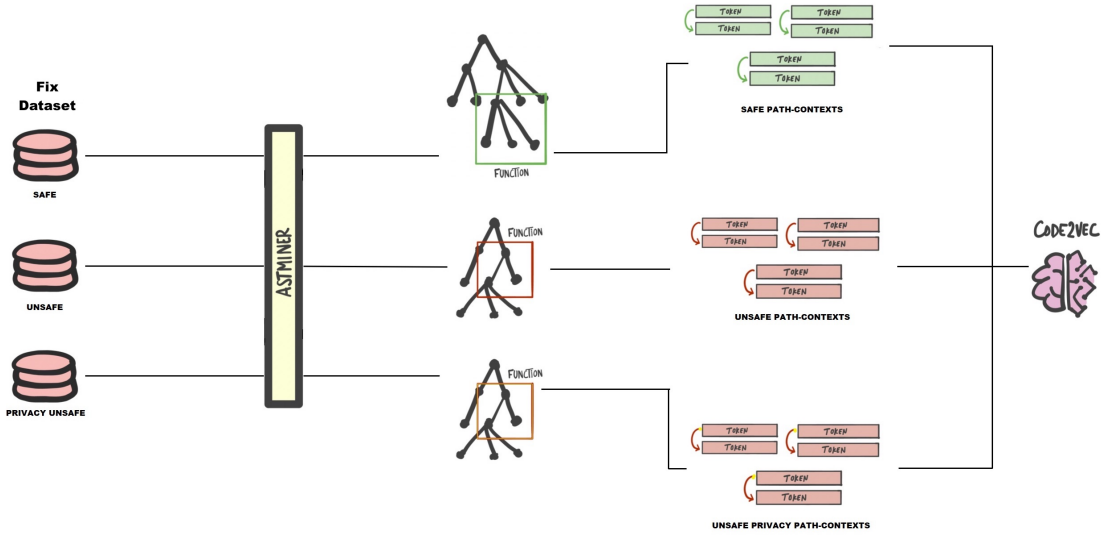

Figura 3.1: Pipeline di machine learning

3.2 Dataset

	Vulnerabili	Privacy Vulnerabili	Non vulnerabili
Test	50	50	120
Valid	50	50	120
Train	383	383	918
Totale	483	483	1158
Totale aggregato	966		1158

Tabella 3.1: Composizione fix dataset

Come abbiamo visto nello stato dell'arte uno degli elementi essenziali per un buon modello di predizione di vulnerabilità basato su ML è il dataset che viene utilizzato per addestrare e poi per testare il modello, questo deve essere ben bilanciato e con dati accurati, inoltre in questo caso deve contenere vulnerabilità legate a problemi di privacy. Per ottenere tutto ciò ho deciso di partire dallo studio CVEfixes Dataset[24], questa ricerca ha messo insieme un database contenente funzioni vulnerabili che sono state individuate in codice sorgente open source e pubblicate nel national vulnerability database. Il database in questione inizialmente contiene dati relativi a 29309 file e ben 98250 funzioni in numerosi linguaggi di programmazione con i relativi commit prefix e postfix. Ho provveduto a filtrare il database per ottenere solo i record che riguardano problemi di Controllo degli accessi interrotto. In particolare "CVEfixes Dataset" è messo a disposizione sotto forma di database relazionale compresso in archivio tar.gz, una volta estratto il file .db da questo archivio (pesa oltre 20GB) ho individuato la tabella "method _change" la quale contiene dati relativi a metodi pre e post fix di metodi che sono risultati vulnerabili, questo è indicato attraverso il campo "before_change", quando è True indica che il codice sorgente contenuto nel campo code è di un commit precedente al fix (quindi risulta vulnerabile), quando invece il valore è False si tratta si un commit post fix ed è quindi codice non vulnerabile, il campo name invece descrive brevemente la vulnerabilità. La mia idea è stata quella di rinominare "before _change" in target e cambiare i valori di questo campo in "Privacy" quando in name è presente la stringa "user" (perchè queste vulnerabilità riguardano l'autenticazione dell'utente all'interno dell'applicazione), in questo modo ho ottenuto 483 metodi con target Privacy. A questo punto per creare i dataset di test, validazione e train ho creato 9 copie dello stesso database: I primi 3 sono stati filtrati per ottenere solo i record vulnerabili, 3 con solo i record non vulnerabili, 3 con record con vulnerabilità Privacy. A questo punto ho fatto in modo da filtrare questi database per avere per esempio nella cartella Privacy il db train con tutti i record dedicati al train con target Privacy e così via.

Successivamente ho esportato tramite il tool SQLite questi dati in formato json e fatto il merge per ottenere i dataset finali, mi sono poi dovuto servire del tool " [25]convertjson" per ottenere il formato jsonl utilizzato da Astminer. Per seguire il suggerimento di [12] e creare un dataset con lo stesso numero di di funzioni vulnerabili e funzioni con vulnerabilità specifica, ho mantenuto solo 483 record di metodi con vulnerabilità generica, i

record non vulnerabili sono invece del 20% maggiori della somma di quelli vulnerabili e privacy vulnerabili, sono poi stati ripartiti in train, test e valid come indicato nella tabella 3.1.

3.3 Astminer

Ottenuto il dataset, ho utilizzato il tool open source Astminer [9] per ricavare dal codice sorgente i cosiddetti AST e i relativi AST path e path-context. Definiti come riportato nel capitolo di background e stato dell'arte.
Astminer [9]: è una libreria per l'estrazione di rappresentazioni del codice basate sui percorsi supportata dal team Machine Learning Methods for Software Engineering presso JetBrains Research.
La libreria è messa a disposizione come tool riusabile e contiene i moduli di filtraggio per eliminare i dati ridondanti, di label extractors per creare etichette per ogni albero e di storages per definire il formato di archiviazione. Un'altra caratteristica fondamentale per la nostra ricerca è la compatibilità con tutti i linguaggi di programmazione, infatti il framework code2vec integra già uno strumento per ricavare i path-context dal codice sorgente ma non è compatibile con il linguaggio C utilizzato dai metodi di fix dataset.
Nella versione utlizzata in dx2021 viene usato uno script personalizzato che visita tutte le righe del dataset in formato jsonl passando in input tutte le funzioni indipendentemente dal fatto che siano vulnerabili o non vulnerabili o nel mio caso privacy vulnerabili, per ogni funzione astminer calcola il suo AST e da questo i relativi path-context che vengono poi etichettati e passati in input a code2vec [10] per il traning del modello di ML. Per adattare l'Astminer della ricerca dx2021 è stato sufficiente modificare leggermente lo script personalizzato realizzato dal team di [1], Il loro dataset chiamato Devign aveva come target un valore binario (0: non vulnerabile e 1: vulnerabile), questi 0 e 1 venivano convertiti nelle stringhe "safe" e "vuln", nel mio caso invece posso avere True, False o Privacy che faccio diventare rispettivamente vuln, safe e priv.

3.4 Code2vec

Code2vec [10] è in grado di aggregare i path-context estratti tramite Astminer catturando somiglianze semantiche, analogie e combinazioni come dimostrato nell'articolo,

assegnando a questi vettori di path-contexts che rappresentano centinaia di espressioni e istruzioni nel corpo di un metodo assegnando un etichetta descrittiva che nel nostro caso può essere "vuln", "priv" o "safe". Principi di funzionamento del framework.
Per fare quanto descritto il framework utilizza una rete neurale (nel campo dell'apprendimento automatico, una rete neurale è un modello computazionale composto di neuroni artificiali ispirati a una rete neurale biologica) che apprende gli incorporamenti di codice in modo da modellare la corrispondenza tra frammenti di codice ed etichette : alla base c'è l'idea che la distribuzione delle etichette possa essere dedotta dai percorsi sintattici del codice c. Il tool è in grado di riconoscere quelle che vengono definite borse invisibili di path-context, questo perchè metodi simili non avranno esattamente lo stesso contenitore di contesti di percorso. Abbiamo quindi bisogno di un meccanismo compositivo che possa aggregare un sacchetto di contesti di percorso in modo tale che i sacchetti che producono la stessa etichetta siano mappati su vettori vicini. Un tale meccanismo compositivo sarebbe in grado di generalizzare e rappresentare nuove borse invisibili utilizzando i singoli percorsi-contesti e le loro componenti (percorsi, valori, ecc.) che sono stati osservati durante l'allenamento come parti di altre borse [10]. Per fare questo viene utilizzata una rete di attenzione neurale soft che apprende quanta attenzione deve essere data a ogni elemento di un sacchetto di path-contexts assegnando dei pesi ai vari componenti. Attenzione soft vuol dire che i pesi vengono distribuiti su tutti i path-context, si potrebbe invece parlare di attenzione hard se ci si concentrasse su un path-context per volta.

3.4.1 Architettura di code2vec

Come riportato in Figura 3.3 il framework utilizza uno strato completamente connesso e uno strato di attenzione neurale soft, agendo in questa maniera:
Nella rete ogni path-context viene rappresentato tramite un context vector, questo è a sua volta composto da 3 vettori : il primo contiene i vocaboli che identificano i nodi sorgente, il secondo è contiene un percorso, il terzo elemento contiene i vocaboli che identificando i nodi terminali. Uno strato completamente connesso impara a combinare questi 3 vettori, ottenendo un cosiddetto vettore di contesto combinato, questo può essere matematicamente descritto come la tangente iperbolica del prodotto tra il context vector e una matrice dei pesi W appresa. Questo significa che un vettore in tre dimensio-

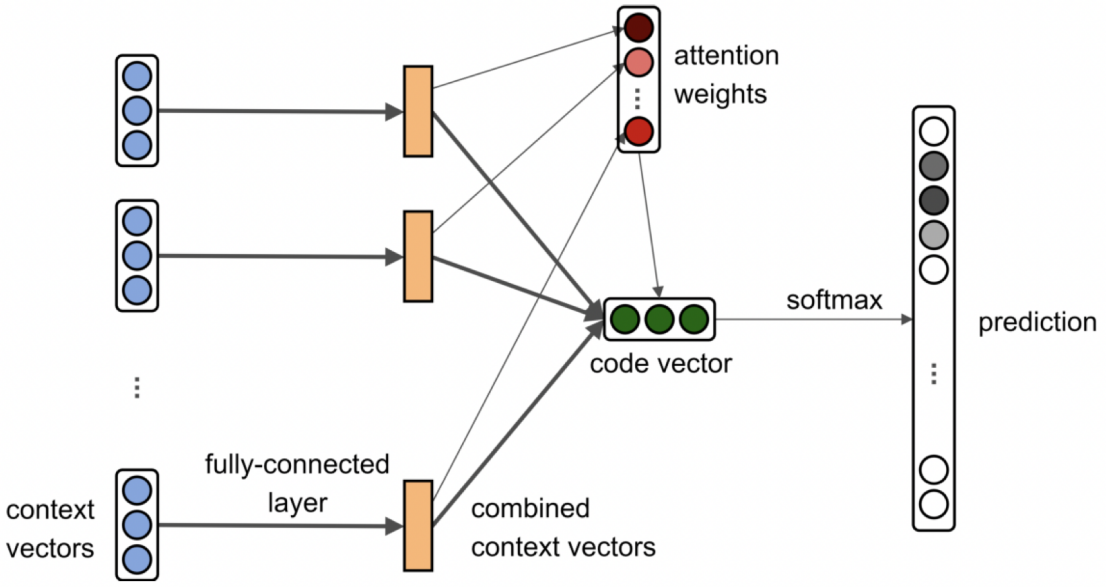

Figura 3.2: Architettura code2vec

ni viene compresso dallo strato completamente connesso in un vettore di dimensione d tramite la moltiplicazione con la matrice W, a ogni elemento viene poi applicata la funzione tanh (genera un valore di output tra -1 e 1 che aumenta l'espressività del modello). Alla fine di questa fase si ottengono quindi quelli che vengono definiti context vector aggregati.

Il meccanismo di attenzione calcola una media ponderata dei context vector aggregati per ottenere un unico vettore, in particolare questa si ottiene ponderando ogni vettore per il suo prodotto scalare con un vettore di attenzione globale. Il vettore di attenzione globale viene calcolato attraverso la tecnica della retropropagazione: diminuire la cosiddetta perdita di entropia incrociata fra la distribuzione prevista e quella esatta fa aumentare la probabilità che il modello assegni true alle etichette vere, quindi si utilizza la retropropagazione dell'errore di addestramento attraverso tutti i parametri appresi derivando la perdita di entropia rispetto ad ogni parametro appreso e aggiornandolo di un piccolo step verso la minimizzazione della perdita. Alla fine di questo processo si ottiene un vettore di codice aggregato.

Inoltre durante tutta la fase di training, il framework apprende un vocabolario dei tag presenti nel corpo di addestramento, generando un vettore di incorporamento dei tag per ogni tag presente.

Infine il vettore di codice aggregato viene utilizzato per la previsione dei tag che sono presenti nel corpo di addestramento, cioè viene fatto il prodotto scalare tra il vettore di

codice e il vettore di incorporamento di tag per un determinato tag.

3.4.2 Configurazione e adattamento del framework

Per utilizzare code2vec per il mio scopo ho dovuto modificare lo script di configurazione per prendere in input i file ottenuti da astminer, preprocessarli assegnando al dataset il nome fix per evitare di utilizzare file del progetto originale e generare il nuovo modello che ho chiamato fixesmodel, come nel progetto originale, per questa versione iniziale, ho utilizzato gli iperparametri di default per code2vec: batch size di 1024, embedding size di 128, dropout rate di 0.75.
Una volta configurato gli iperparametri, il framework è in grado di generare una serie di versioni del nostro modello. Per valutare quale sia il modello migliore generato, per ogni versione viene generata una predizione su un determinato dataset (nel mio caso sul dataset valid). Il modello più promettente ottenuto in questa fase portava ai risultati preliminari della Tabella 3.2. Tutte le tabelle con la dicitura "risultati preliminari" si riferiscono alle metriche di valutazione ottenute in fase si training eseguendo la predizione delle istanze contenute nel file "valid" del fix dataset.

	Precision	Accuracy	Recall	F1
Valid	0,500	0,546	0,082	0,140

Tabella 3.2: fixmodel0 risultati preliminari

Questi dati risultano interessanti per alcuni parametri come la precisione ma mostrano alcuni punti deboli come un F1 migliorabile. Per migliorare i risultati preliminari e ottenere modelli più robusti ho deciso di lavorare sugli iperparametri.

3.5 Ottimizzazione degli iperparametri

La prima cosa che ho notato è stato il parametro NUM_TRAIN_EPOCHS impostato a 20, aumentando progressivamente il parametro ho scoperto che il framework è in grado di generare un numero maggiore di versioni e questo aumenta inevitabilmente la probabilità di trovare modelli interessanti, infatti tra le versioni generate, l'iterazione numero 29 porta questi risultati preliminari della Tabella 3.3.

	Precision	Accuracy	Recall	F1
Valid	0,541	0,569	0,337	0,415

Tabella 3.3: fixmodel0.1 (epoch29) risultati preliminari

Come si può vedere c'è un miglioramento tangibile per le metriche F1 e Recall. Inoltre, ho scoperto che il numero massimo di versioni generabili con il dataset di training utilizzato è di 150 ma dopo tale iterazione il test si blocca automaticamente, ho quindi deciso di impostare a 200 il parametro per i test successivi.

Il secondo parametro che ha catturato la mia attenzione è MAX_PATH_VOCAB_SIZE, aumentare il valore di tale parametro significa aumentare la dimensione massima del vocabolario dei percorsi, questo secondo la ricerca [1] non dovrebbe far aumentare necessariamente le prestazioni del modello. Ho deciso di testare ciò raddoppiando il valore che è passato da 911417 a 1822834 ottenendo questi risultati i risultati preliminari in Tabella 3.4.

	Precision	Accuracy	Recall	F1
Valid	0,512	0,551	0,244	0,312

Tabella 3.4: fixmodel0.2 risultati preliminari

Come previsto dalla ricerca il modello generato non è migliore di quelli ottenuti fino a questo punto, ho deciso quindi di rimanere invariato il parametro e proseguire.

Il terzo parametro sul quale ho deciso di agire è TRAIN_BATCH_SIZE. Questo parametro definisce quanti campioni del dataset di addestramento vengono propagati sulla rete ad ogni iterazioni. Un TRAIN_BATCH_SIZE più elevato può portare a una stima più precisa del gradiente (descrive come la funzione possa variare in base ai suoi parametri) della funzione finale che verrà ottenuta alla fine della fase di addestramento. Raddoppiando il valore di tale parametro sono riuscito ad ottenere un modello molto interessante in Tabella 3.5.

	Precision	Accuracy	Recall	F1
Valid	0,741	0,606	0,204	0,320

Tabella 3.5: fixmodel0.3 risultati preliminari

Per verificare se si trattasse di una casualità ho deciso di decrementare lo stesso parametro portandolo alla metà del valore originale, i risultati preliminari ottenuti con questa variazione sono riportati nella Tabella 3.6.

	Precision	Accuracy	Recall	F1
Valid	0,500	0,546	0,214	0,300

Tabella 3.6: fixmodel0.4 risultati preliminari

Come si può notare si ottengono i peggiori risultati preliminari fino a questo punto. Ho provato ad aumentare ulteriormente il valore ma questo andava oltre le risorse hardware di test: il processo occupava troppa memoria di sistema e veniva killato dal sistema operativo, ho dovuto mantenere quindi il valore a 2048.

L'ultimo iperparametro sul quale ho agito è il DROPOUT_KEEP_RATE.
Il DROPOUT_KEEP_RATE è il parametro che definisce la probabilità che un nodo della rete possa essere eliminato, un valore più elevato può diminuire i problemi di overfitting, rendendo il modello generato meno legato al dataset di addestramento. Nella versione originale di code2vec questo parametro era impostato a 0,25 a differenza del 0,75 di dx2021[1], ho deciso di riportarlo a tale valore ottenendo un modello molto interessante come in Tabella 3.7.

	Precision	Accuracy	Recall	F1
Valid	0,750	0,630	0,276	0,403

Tabella 3.7: fixmodel0.5 risultati preliminari

3.6 Generazione dei modelli finali

Alla fine del training sono state create le release di tutti i modelli illustrati sopra, le release sono poi state usate per generare le predizioni dei vari dataset (di training, di validazione e di test).

3.7 Preparazione per il confronto con dx2021

Ho deciso infine di mettere a paragone i modelli generati in questa ricerca con quelli originali prodotti in dx2021 [1], per fare ciò ho utilizzato i file originali del progetto e seguito le istruzioni riportate sulla pagina GitHub del progetto, successivamente ho utilizzato il miglior modello così ottenuto per predire il dataset fix ricavato come descritto in questa ricerca.
Il modello è stato testato sul dataset di validazione di Devign[26] ottenendo i risultati preliminari in Tabella 3.8.

	Precision	Accuracy	Recall	F1
Valid(Devign)	0,571	0,622	0,524	0,547

Tabella 3.8: dx2021model risultati preliminari

Per rendere compatibile il modello ottenuto con fix dataset, ho dovuto modificare nuovamente uno degli script di Astminer[9] per trasformare tutte le etichette "Privacy" in "vuln" in modo da adattare i path-context generati al modello originale ottenuto.

CAPITOLO 4

Risultati

In questo capitolo saranno illustrati i risultati ottenuti dai vari modelli mostrati nel capitolo precedente che sono stati ottenuti tramite diverse configurazioni degli iperparametri di code2vec. Tutte le tabelle con la dicitura "risultati" riportano le metriche di valutazione ottenute dalle varie release del modello quando è utilizzato per la predizione dei metodi contenuti nei file (train, valid e test) del fix dataset.

4.1 Modelli finali

Il modello originale (senza variazione degli iperparametri) conferma buoni risultati su tutte le metriche di valutazione, generare più modelli permette però di migliorare le prestazione su quasi tutte le metriche e per ogni dataset, fixmodel0.1 viene però superato per Precision e Accuracy sul dataset di test. Rispettivamente tabelle 4.1 e 4.2. Considerare un vocabolario dei percorsi di dimensione maggiore si conferma invece peggiorativo per le prestazioni: tabella 4.3. Raddoppiare il parametro TRAIN_BATCH_SIZE porta a un modello con elevate prestazioni che però soffre sulla metrica di Recall, dimezzare invece il valore dello stesso parametro porta invece a peggiorare le prestazioni tranne proprio per il Recall. Rispettivamente tabelle 4.4 e 4.5.
Il modello generato riportando il valore del parametro DROPOUT_KEEP_RATE al valore di 0,25 (come previsto originariamente in code2vec), come riporta la Tabella 4.6, porta ottime prestazioni in Valid e Test. Per il dataset di Test viene invece sconfitto per F1 e Recall da fixmodel0 e 0.1. Per Precision e Accuracy viene invece sconfitto da fixmodel0.3. Curiosamente le prestazioni risultano invece inferiori sul dataset di training, questo è probabilmente dovuto al fatto che il parametro impostato in questo modo faccia ottenere i migliori risultati su Valid nelle prime iterazioni, questo porta a un basso addestramento sul dataset di Train, infatti fixmodel0.5 è stato generato dopo poche epochs.

	Precision	Accuracy	Recall	F1
Train	0,982	0,763	0,431	0,599
Valid	0,500	0,546	0,082	0,140
Test	0,345	0,514	0,104	0,16

Tabella 4.1: fixmodel0 risultati

	Precision	Accuracy	Recall	F1
Train	0,892	0,877	0,799	0,843
Valid	0,541	0,569	0,337	0,415
Test	0,193	0,245	0,219	0,205

Tabella 4.2: fixmodel0.1 (epoch29) risultati

	Precision	Accuracy	Recall	F1
Train	0,881	0,859	0,760	0,816
Valid	0,512	0,551	0,244	0,312
Test	0,135	0,231	0,135	0,135

Tabella 4.3: fixmodel0.2 (epoch30) risultati

4.2 Confronto con dx2021model

Il modello di confronto che è stato utilizzato per predire il dataset fixes nelle sue tre componenti opportunamente adattate (come descritto nel capitolo 3), ha ottenuto i risultati in tabella 4.7.

Il modello di confronto generato come in dx2021[1], come si può vedere dalla Tabella 4.7, nonostante abbia dovuto assegnare soltanto due etichette (safe e vuln), evidenzia risultati inferiori alla maggioranza dei modelli generati in questa ricerca. In particolare viene battuto in tutte le metriche da fixmodel0.3.
dx2021model riesce comunque a rimanere molto stabile sulla metriche di Accuracy che per il dataset di Test risulta tra le più alte, superata solo da fixmodel0.3 e fixmodel0.

	Precision	Accuracy	Recall	F1
Train	0,919	0,622	0,089	0,162
Valid	0,741	0,606	0,204	0,320
Test	0,481	0,551	0,135	0,211

Tabella 4.4: fixmodel0.3 risultati

	Precision	Accuracy	Recall	F1
Train	0,898	0,913	0,889	0,893
Valid	0,500	0,546	0,214	0,300
Test	0,147	0,222	0,156	0,152

Tabella 4.5: fixmodel0.4 risultati

	Precision	Accuracy	Recall	F1
Train	0,554	0,609	0,256	0,350
Valid	0,750	0,630	0,276	0,403
Test	0,400	0,166	0,167	0,235

Tabella 4.6: fixmodel0.5 risultati

	Precision	Accuracy	Recall	F1
Train	0,479	0,585	0,108	0,176
Valid	0,294	0,523	0,052	0,088
Test	0,294	0,523	0,052	0,088

Tabella 4.7: dx2021model risultati

CAPITOLO 5

Conclusioni

5.1 Analisi dei risultati

I risultati ottenuti mostrano che l'addestramento di modelli di predizione di vulnerabilità basato sull'analisi statica del codice sorgente e in grado di classificarle come rischio per la privacy, in questo caso individuando quelle relative al Controllo degli accessi interrotto, è una strada molto promettente. La preparazione di un dataset bilanciato a tale scopo porta inevitabilmente a ridurre il numero di istanze per l'addestramento, pertanto il modello può avere prestazioni variabili a seconda del dataset di test. I migliori risultati sono stati ottenuti dal modello fixmodel0.3 ricavato utilizzando un iperparametro di TRAIN_BATCH_SIZE di 2048 e fissando il parametro di NUM_TRAIN_EPOCHS a 200.
Il modello di confronto generato (dx2021model), che ottiene ottimi risultati sul dataset Devign[26] come dimostrato in dx2021[1], mostra prestazioni inferiori a fixmodel0.3 su tutte le metriche quando utilizzato per etichettare le istanze di fixes come vulnerabili o non vulnerabili.

5.2 Riflessione sui risultati complessivi

Il lavoro effettuato ha portato a ricavare modelli con ottime prestazioni sulla maggioranza delle metriche di valutazione ma si potrebbe ampliare la mole di dati per l'addestramento relative a vulnerabilità di Controllo degli accessi interrotto, in modo da provare a definire un modello ancora più robusto.

I risultati indicano che non necessariamente l'aumento del numero di etichette da assegnare e quindi una classificazione a grana più fine, faccia peggiorare le prestazioni della pipeline di machine leraning. Al contrario nel caso di questa ricerca, grazie a un addestramento adeguato e all'ottimizzazione degli iperparametri è stato possibile ottenere prestazioni maggiori. Quindi sarebbe importante allargare la ricerca ad altri tipi di vulnerabilità che comportano un rischio per la privacy e accorpare le varie ricerche compresa la mia e [12]. In modo da definire uno strumento unico in grado di individuare tutti i tipi di vulnerabilità per la privacy definiti in [11], provando ad assegnare un'etichetta diversa per ogni vulnerabilità. In questo modo potremmo capire se l'aumento ulteriore delle etichette faccia degenerare le prestazioni o se queste possano essere mantenute.

Ringraziamenti

A conclusione di questo elaborato, vorrei dedicare qualche riga a tutti coloro che mi sono stati vicini in questo percorso di crescita personale e professionale.

Innanzitutto, ringrazio il prof. Fabio Palomba, il dott. Giammaria Giordano e tutto il team del SESA LAB che in questi mesi mi hanno supportato nella stesura dell'elaborato. Grazie a loro ho accresciuto le mie conoscenze e le mie competenze e sono state fonte di ispirazione per la trascrizione di questa tesi.

Ringrazio la mia famiglia per avermi sostenuto in questo percorso.

Ringrazio i colleghi e amici: Maddalena, Gaetano e Vincenzo. Con loro ho passato tanti momenti di crescita durante il progetto di ingegneria del software.

Ringrazio i miei amici Gaetano, Arturo, Pasquale, Antonio, Alfonso, Luigi, Francesco, Emanuel, Christian, Raffaele, Angelo, Dario e Giuseppe per il supporto emotivo in questi anni.

Infine, ringrazio me stesso per la perseveranza che mi ha fatto superare i momenti più difficili.

Bibliografia

[1] R. A. C. P. H. E. David Coimbra, Sofia Reis, "On using distributed representations of source code for the detection of c security vulnerabilities," *arxiv.org*, 2021. (Citato alle pagine 5, 11, 17, 18, 20, 24, 25, 26, 29 e 31)

[2] R. Flor, "Lotta alla "criminalità informatica" e tutela di "tradizionali" e "nuovi" diritti fondamentali nell'era di internet*," *penalecontemporaneo.it*, 2022. (Citato a pagina 1)

[3] cybersecurity360.it, "Facebook, pubblici i dati di 36 milioni di italiani: come possiamo difenderci," *cybersecurity360.it*, 2022. (Citato a pagina 2)

[4] cybersecurity360.it, "Web scraping, tutto sulla tecnica usata per rubare dati facebook e linkedin," *cybersecurity360.it*, 2022.

[5] swisscom.ch, "Cosa dovrebbero sapere le pmi su furti di dati e darknet," *swisscom.ch*, 2022.

[6] clusit.it, "Rapporto clusit 2022 – edizione di marzo 2022," *clusit.it*, 2022. (Citato a pagina 2)

[7] gazzettaufficiale.it, "Disposizioni urgenti in materia di cybersicurezza, definizione dell'architettura nazionale di cybersicurezza e istituzione dell'agenzia per la cybersicurezza nazionale. (21g00098)," *gazzettaufficiale.it*, 2022. (Citato a pagina 2)

[8] agendadigitale.eu, "Crescono gli attacchi cyber in italia, ma anche le difese: ecco il quadro," *agendadigitale.eu*, 2022. (Citato a pagina 3)

[9] V. Kovalenko, E. Bogomolov, T. Bryksin, and A. Bacchelli, "Pathminer: a library for mining of path-based representations of code," pp. 13–17, 2019. (Citato alle pagine 5, 6, 17, 20 e 26)

[10] O. L. E. Y. Uri Alon, Meital Zilberstein, "code2vec: Learning distributed representations of code," *doi.org*, 2019. (Citato alle pagine 5, 6, 17, 18, 20 e 21)

[11] www.reflectiz.com, "A complete review of the owasp top ten for 2022," *www.reflectiz.com*, 2022. (Citato alle pagine 6, 7, 8, 9, 10, 17 e 32)

[12] P. R. H. Tiago Baptista, Nuno Oliveira, "Using machine learning for vulnerability detection and classification," *drops.dagstuhl.de*, 2021. (Citato alle pagine 6, 8, 15, 16, 17, 19 e 32)

[13] www.softcomet.com, "Le 20 vulnerabilitÀ piÙ significative che hanno paralizzato il mondo del web (it) nel decennio 2010-2020," *www.softcomet.com*, 2022. (Citato a pagina 10)

[14] wikipedia, "Heartbleed," *wikipedia*, 2022. (Citato a pagina 10)

[15] B. L. L. S. Z. C. M. Li, "Software vulnerability discovery techniques: A survey," *IEEE*, 2013. (Citato alle pagine 12 e 13)

[16] P. Louridas, "Static code analysis," *IEEE*, 2006. (Citato a pagina 12)

[17] M. Bishop, "About penetration testing," *IEEE*, 2007. (Citato a pagina 13)

[18] A. Hewko, "Stride threat modeling: What you need to know," *softwaresecured.com*, 2021. (Citato a pagina 13)

[19] D. Berardo, "Tecniche owasp per la qualità del codice: cosa sono e chi deve applicarle, alla luce del gdpr," *clusit.it*, 2020. (Citato a pagina 13)

[20] tecnoteca, "Osstmm," *tecnoteca*, 2022. (Citato a pagina 13)

[21] T. M. I. P. F. Massacci, "Machine learning for source code vulnerability detection: What works and what isn't there yet," *IEEE*, 2022. (Citato a pagina 14)

[22] R. R. L. K. L. H. T. L. J. H. O. O. P. E. M. McConley, "Automated vulnerability detection in source code using deep representation learning," *IEEE*, 2018. (Citato a pagina 15)

[23] F. W. J. W. J. L. W. Wang, "Vulnerability detection with deep learning," *IEEE*, 2017. (Citato a pagina 15)

[24] G. Bhandari, A. Naseer, and L. Moonen, "Cvefixes: Automated collection of vulnerabilities and their fixes from open-source software," *ACM*, p. 10, 2021. (Citato alle pagine 6, 17 e 19)

[25] nd, "convertjson," *convertjson.com*, 2018. (Citato a pagina 19)

[26] J. S. X. D. Y. L. Yaqin Zhou, Shangqing Liu, "Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks," *arxiv.org*, 2019. (Citato alle pagine 26 e 31)