Capitolo 1

Introduzione

Negli ultimi anni, gli Smart Contracts hanno acquisito una crescente popolarità, attirando l'attenzione di aziende e sviluppatori grazie alle loro caratteristiche innovative, come l'esecuzione automatica, la trasparenza, l'immutabilità e la decentralizzazione. Gli Smart Contracts sono programmi che eseguono automaticamente azioni al verificarsi di condizioni predefinite, operando su una blockchain, un registro distribuito che mantiene una lista di record, chiamati blocchi, collegati in modo sicuro mediante crittografia.

L'attenzione che gli Smart Contracts stanno generando viene confermata dal loro inserimento all'interno di una proposta di regolamentazione da parte dell'Unione Europea [**?**], in cui gli Smart Contracts vengono definiti come:

> "programmi informatici su registri elettronici che eseguono e regolano transazioni sulla base di condizioni prestabilite. Tali programmi possono potenzialmente fornire ai titolari e ai destinatari dei dati garanzie del rispetto delle condizioni per la condivisione dei dati."

La più grande piattaforma di Blockchain che permette l'esecuzione degli Smart Contracts è Ethereum. Gli Smart Contracts di Ethereum sono scritti in Solidity, un linguaggio di programmazione Turing-completo ad alto livello. Tuttavia, la scrittura di Smart Contracts sicuri rappresenta una sfida, in quanto anche piccoli errori possono portare a gravi conseguenze. Gli Smart Contracts sono immutabili, di conseguenza una volta pubblicati non possono essere modificati. Questa loro caratteristica, per quanto rappresenti una conferma del fatto che il contratto accettato dalle parti non possa essere modificato, può rilevarsi il loro principale punto debole. Difatti, qualsiasi errore o vulnerabilità presente in uno Smart Contract non può essere corretto, dando la possibilità a soggetti malintenzionati di approfittarne per rubare fondi o causare altri danni.

Diventa, di conseguenza, fondamentale per gli sviluppatori disporre di tool che permettano di rilevare automaticamente le vulnerabilità presenti negli Smart Contracts che

sviluppano, assicurandosi che siano privi di potenziali problemi prima della pubblicazione del contratto. Attualmente, per risolvere questo potenziale problema si fa riferimento a tecniche di analisi statica, che permettono di analizzare il codice sorgente o il bytecode di uno Smart Contract senza effettuarne l'esecuzione effettiva. Questo approccio consente di identificare potenziali problematiche senza la necessità di testare il codice in un ambiente reale, di contro però queste tecniche si rivelano essere molto lente e fanno troppo affidamento al riconoscimento di pattern conosciuti o si basano su dei set di regole predefinite, che possono non essere esaustive o non riconoscere nuove vulnerabilità.

Un'alternativa a queste tecniche è l'utilizzo di tecniche di Machine Learning e Deep Learning. Queste tecniche permettono di rilevare le vulnerabilità presenti negli Smart Contracts in maniera più rapida e con una maggiore accuratezza rispetto alle tecniche di analisi statica. In questo campo, negli ultimi anni c'è stato un notevole interesse e sono stati proposti diversi approcci che fanno uso di queste tecniche. Questi approcci si basano sull'addestramento di modelli di Machine Learning e Deep Learning, spaziando da modelli basati su modelli di Machine Learning più tradizionali a modelli deep basati su reti neurali convoluzionali.

Questo lavoro di tesi si inserisce in questo contesto e propone un approccio basato su tecniche di Deep Learning utilizzando modelli basati sui Transformers per la rilevazione di vulnerabilità negli Smart Contracts. In particolare, questo lavoro ha l'obiettivo di costruire dei modelli che siano in grado di rilevare cinque classi di vulnerabilità:

- Access-Control
- Arithmetic
- Other
- Reentrancy
- Unchecked-Calls

Il contributo di questo lavoro mira a dimostrare l'efficacia dei modelli basati sui Transformers nella rilevazione di vulnerabilità all'interno degli Smart Contracts. In particolare, sono stati impiegati i modelli BERT, DistilBERT e CodeBERT per la classificazione multilabel, poichè un singolo Smart Contract può presentare simultaneamente diversi tipi di vulnerabilità. Considerando che questi modelli hanno un limite di 512 token per input e che i contratti possono essere significativamente più lunghi, è stato necessario adottare un approccio alternativo. Studi precedenti hanno dimostrato che suddividere i contratti in sottocontratti e classificarli mantenendo la stessa etichetta porta alla non convergenza della loss del modello [**?**]. Pertanto, è stato sperimentato un metodo che prevede la segmentazione del testo in sottocontratti, seguita dall'aggregazione o concatenazione degli embedding generati per ciascun sottocontratto. Questi modelli sono stati addestrati sia utilizzando il bytecode che il codice sorgente dei contratti. Una volta

ottenuti il miglior modello per ognuna delle modalità dei dati in input (codice sorgente e bytecode), è stato utilizzato un modello ensamble per ottenere un modello che effettua le predizioni a partire dalle predizioni dei migliori modelli addestrati rispettivamente sul codice sorgente e sul bytecode. Infine, data la crescente diffusione di chatbot e assistenti virtuali nelle nostre vite, sia per il supporto a compiti di vario genere sia come ausilio alla programmazione, è stato testato il modello Gemini di Google per valutare la sua performance in un task di classificazione di questo tipo.

I risultati ottenuti dimostrano che i modelli basati sui Transformers sono altamente efficaci nel rilevare le vulnerabilità negli Smart Contracts. Tra questi, CodeBERT, un modello preaddestrato su codice sorgente, ha mostrato una notevole capacità di identificare le vulnerabilità sia nel codice sorgente che nel bytecode esadecimale. I modelli addestrati utilizzando diverse tecniche di aggregazione del testo hanno ottenuto performance superiori, evidenziando l'importanza di considerare l'interezza del contratto. Inoltre, l'uso di meta-classificatori in un modello ensemble ha ulteriormente migliorato le performance, dimostrando l'efficacia delle tecniche di combinazione dei modelli. In contrasto, il modello Gemini di Google, ha ottenuto risultati non soddisfacenti, sottolineando la necessità di utilizzare modelli specializzati per la rilevazione di vulnerabilità negli Smart Contracts.

La suddivisione di questo lavoro è articolata come segue:

- il capitolo due illustra le motivazioni alla base del presente lavoro di tesi, introducendo il concetto di Smart Contract e sottolineando l'importanza della rilevazione delle vulnerabilità in essi presenti. Verranno inoltre descritte alcune delle principali classi di vulnerabilità riscontrabili negli Smart Contract. Successivamente, sarà presentata una revisione della letteratura esistente in questo campo, con una panoramica dei lavori più significativi che hanno trattato questo tema.

- il terzo capitolo espone la metodologia adottata in questo lavoro. Inizialmente, sarà condotta un'analisi esplorativa del dataset, al fine di estrarne informazioni utili per la costruzione dei modelli. Successivamente, saranno introdotti i modelli utilizzati, BERT, DistilBERT e CodeBERT, con una dettagliata descrizione delle loro architetture e delle modalità di addestramento. Saranno illustrate le implementazioni e il setup dei vari esperimenti condotti. Inoltre, in questo capitolo sarà presentato il modello Gemini di Google e verrà spiegato come è stato applicato nel contesto di questo studio.

- il quarto capitolo riporta i risultati ottenuti nel corso del lavoro. Verranno introdotte le metriche di valutazione utilizzate e saranno presentate le performance dei vari modelli addestrati. Infine, saranno mostrati i risultati ottenuti dai meta-classificatori e dal modello Gemini di Google.

- il quinto ed ultimo capitolo riporta le conclusioni del lavoro svolto, con una discussione delle possibili direzioni future di ricerca.