Capitolo 1

Conclusioni e sviluppi futuri

In questo lavoro di tesi, sono stati utilizzati dei modelli di Deep Learning basati sui Transformers per la classificazione delle vulnerabilità negli Smart Contracts. Sono stati inizialmente introdotti gli Smart Contracts e le tecnologie correlate, come la blockchain, delineando il loro funzionamento i principali vantaggi che hanno portato. In seguito, è stata effettuata un'analisi delle vulnerabilità più comuni che affliggono gli Smart Contracts, soffermandosi sopratutto sulle vulnerabilità prese in considerazione durante la fase sperimentale di questo lavoro. Successivamente, è stata esaminata anche la letteratura presente e le metodologie di classificazione attuali.

Il focus si è, poi, spostato sulla presentazione dei modelli utilizzati in questo studio, in particolare sui modelli della famiglia BERT basati sull'architettura dei Transformers. Tra le principali contribuzioni di questo lavoro vi è la dimostrazione della maggior efficacia dei modelli CodeBERT, preaddestrati su codice sorgente, rispetto ai modelli BERT pre-addestrati su testi generici. Inoltre, sono state testate diverse metodologie che hanno permesso di superare il limite di lunghezza di sequenze di token in input ai modelli BERT, consentendo di analizzare interi (o quasi) Smart Contracts. Nello specifico, il codice dei contratti è stato suddiviso in blocchi da 512 token, e sono stati utilizzati approcci in cui gli embedding prodotti dal modello per ciascun blocco sono stati concatenati o aggregati. Questi approcci di concatenazione degli embedding sono stati scelti poichè precedenti esperimenti effettuati in letteratura hanno dimostrato come approcci più classici di Sliding Window sul testo o di segmentazione in sottoparti e ricongiunzione delle predizioni peggiorino le performance dei modelli. Infine, è stato presentato un approccio di stacking, che ha permesso di combinare il miglior modello preaddestrato sul codice sorgente ed il miglior modello preaddestrato sul bytecode degli Smart Contracts. Nello stacking un meta-classificatore ha il compito di predire la classificazione finale a partire dalle predizioni dei classificatori base. Sono stati testati vari algoritmi di machine learning utilizzati come meta-classificatori con l'obiettivo di migliorare le performance dei modelli base. I risultati ottenuti hanno dimostrato come l'approccio di stacking permetta di ottenere risultati migliori rispetto all'utilizzo di un singolo modello, ma sia facilmente

prono all'overfitting dei modelli. Nello specifico, i risultati ottenuti dimostrano come modelli più complessi come SVM e Random Forest siano più soggetti all'overfitting rispetto a modelli più semplici come il Gaussian Naive Bayes. La regressione logistica ha ottenuto ottimi risultati in termini di accuratezza, F1 score e recall senza mostrare segni di overfitting, risultando quindi il miglior modello scelto, anche se modelli come Random Forest, che tendevano a overfittare, hanno ottenuto risultati leggermente migliori.

I risultati di questo lavoro hanno dapprima dimostrato come modelli preaddestrati su codice sorgente, come CodeBERT, ottengano risultati migliori rispetto ai modelli preaddestrati su testi generici. Si è dimostrato, inoltre, come la quantità di testo che viene analizzata sia un fattore chiave nel migliorare le performance dei modelli. I risultati hanno mostrato, però, come il passaggio da due a tre blocchi di codice (quindi il passaggio da 1024 a 1536 token in input), migliori significativamente le performance del modello solo nei modelli che utilizzano il codice sorgente. Questo può essere spiegato sicuramente dalle differenze di lunghezza media tra i due tipi di testo, che fanno si che per il bytecode (in media molto più lungo del codice sorgente) il passaggio da due a tre blocchi non porti a miglioramenti significativi. In termini di risultati ottenuti, sia per il bytecode che per il codice sorgente, il modello che ha ottenuto i risultati migliori è stato il modello che utilizza tre blocchi di codice concatenati con un'accuratezza del 79,39% ed un F1-Score dell'88,24% nel caso del codice sorgente ed un'accuratezza del 76.80% ed un F1-Score dell'86,43% per il modello addestrato sul bytecode. Le migliori metriche di recall sono state ottenute, invece, dai modelli che utilizzano un'aggregazione con funzione max di tre blocchi di codice, con valori di Micro Recall pari all'84% e all'86% rispettivamente per il bytecode e per il codice sorgente. Il modello di stacking con Regressione Logistica ha raggiunto un'accuratezza dell'81,30% ed un F1-Score del quasi 90%. Inoltre, le metriche di recall tutte superiori all'85%, a meno della classe access-control (la classe minoritaria del dataset) che si attesta comunque all'82%, mostrano come il modello sia in grado di classificare correttamente la maggior parte delle vulnerabilità presenti nei contratti, dimostrando come questo approccio, se addestrato su ottimi modelli base, possa essere molto efficace.

Per valutare la necessità di utilizzare modelli ad hoc per task complessi come questo, rispetto ai modelli LLM (Large Language Models) come GPT di OpenAI o Gemini di Google entrati ormai nelle nostre vite sotto forma di chatbot utilizzatissimi per tantissimi task, sono stati effettuati test comparativi. Utilizzando la versione gratuita delle API di Gemini, è stato raccolto un campione di 1169 classificazioni dal dataset di test. I risultati ottenuti non sono soddisfacenti, con un'accuratezza del 27% e delle recall pari quasi allo zero in tutte le classi a meno della classe Other, che raggiunge livelli di recall poco piu bassa del 50%. Questi risultati dimostrano come per task complessi come questo sia ancora necessario utilizzare modelli ad hoc.

Possibili direzioni future di ricerca di questo lavoro implicano sicuramente un miglior tuning dei modelli, sopratutto per quanto riguarda la Batch Size. A seguito, infatti, della limitata potenza di calcolo non è stato possibile tunare correttamente la batch

size, che in molti esperimenti è stata obbligata ad essere impostata a valori molto bassi. Sarebbe inoltre, molto interessate utilizzare modelli di classificazione che possano gestire sequenze di token di lunghezza maggiore come Big Bird [**?**] e LongFormer [**?**]. Allo stesso tempo, avendo a disposizione più risorse computazionali, sarebbe interessante provare a fare finetuning su dei modelli LLM divenuti lo stato dell'arte al giorno d'oggi come GPT, LLaMa o Mistral. Durante lo svolgimento di questo lavoro, è stato già tentato un approccio di fine-tuning di questi modelli, ma la limitata potenza di calcolo non ha permesso di avere tempi di training ragionevoli con una lunghezza delle sequenze di token accettabile. Altre possibili direzioni di ricerca potrebbero vedere l'estensione della classificazione ad altre classi di vulnerabilità, non presenti nel dataset utilizzato in questo studio.