tinexta
defence

Applicazione di Modelli Linguistici a Tecnologie di Deception ad Alta Interazione

#TinextaDefenceBusiness

AI4Cyber

AI4Cyber studio, è il nuovo spazio di ricerca applicata di Tinexta Defence, dedicato alla ricerca e applicazione dell'**Intelligenza Artificiale in ambito cybersecurity**.

Al suo interno, diversi specialisti con competenze ed esperienze eterogenee collaborano per affrontare le sfide poste dal panorama delle minacce informatiche, accompagnare i clienti nell'adozione dell'AI nei propri processi aziendali e condividere le attività di ricerca con la comunità tecnico-scientifica.

Il team cura l'**intero ciclo di sviluppo dei sistemi basati su AI**: dalla raccolta e dal preprocessing dei dati, all'addestramento e validazione dei modelli, fino al deployment in produzione.

Le soluzioni proposte si fondano su tecnologie avanzate di **Machine Learning**, **Deep Learning** e **Large Language Models**, e possono essere personalizzate in base alle specifiche esigenze del cliente.

L'AI Team è costantemente impegnato in attività di ricerca e sperimentazione, con un'attenzione particolare agli **aspetti etici**, alla **trasparenza** e alla **tutela della privacy**.

L'obiettivo è proporre soluzioni innovative che siano in linea con i principi di **equità, inclusività e rispetto dei diritti fondamentali**.

Summary

Abstract

Questo articolo presenta un framework innovativo per la creazione di un honeypot ad alta interazione basato sull'utilizzo di Large Language Model (LLM) in locale. Il sistema simula un terminale Linux ed è in grado di rispondere ai comandi degli attaccanti in modo convincente, tracciando al contempo le loro attività. L'architettura si compone di tre elementi principali: Terminal Protocol Proxy, Prompt Manager e LLM.

È stato implementato un algoritmo di pruning per gestire le limitazioni di contesto intrinseche degli LLM classificando i comandi basati su un valore "F" per determinare il loro impatto sul sistema. L'implementazione ha richiesto un attento processo di prompt engineering e la gestione statica di alcuni comandi critici per superare le inconsistenze degli LLM.

Inoltre, test comparativi su diversi LLM (Mistral, Gemma2, Phi4, Llama3.1, Qwen2.5) e sulle versioni più avanzate come ChatGPT e Claude AI dimostrano l'efficacia dell'approccio proposto.

Autori:

- Simona Sorgente: Team Leader AI4Cyber
- Antonio Addabbo: Analista AI4Cyber

1. Introduzione

Gli honeypot rappresentano uno strumento fondamentale nella sicurezza informatica, consentendo di identificare, monitorare e analizzare potenziali attacchi senza mettere a rischio i sistemi reali. Tradizionalmente, gli honeypot si distinguono in due categorie principali: a bassa interazione (emulano alcuni protocolli) e ad alta interazione (sistemi reali ma isolati e monitorati).

L'avvento degli LLM ha aperto nuove frontiere nel campo della simulazione di sistemi complessi. Il sistema proposto, dunque, rappresenta un approccio innovativo che sfrutta le capacità degli LLM per creare honeypot ad alta interazione capaci di simulare realisticamente un terminale Linux, rispondendo in modo appropriato ai comandi degli attaccanti e monitorando le loro attività.

2. Architettura del sistema

Il sistema ha un'architettura modulare composta da tre componenti principali:

- **Terminal Protocol Proxy:** interfaccia di comunicazione con l'attaccante che svolge diverse funzioni, tra cui la gestione delle connessioni in ingresso, l'incapsulamento dei messaggi ricevuti, l'inoltro dei comandi al Prompt Manager, la ricezione delle risposte dal Prompt Manager e la restituzione delle risposte dell'honeypot all'attaccante.

- **Prompt Manager:** è il fulcro del sistema e di conseguenza la parte su cui si concentra il progetto. Infatti, si occupa di interagire con lo specifico LLM, di estrarre le risposte generate da quest'ultimo, di inoltrarle al Terminal Protocol Proxy, di gestire la memoria del sistema e lo stato delle interazioni.

- **LLM:** è il motore di generazione delle risposte. Riceve i prompt costruiti dal Prompt Manager, elabora le richieste e genera risposte che simulano quelle di un reale terminale Linux. Per l'implementazione sono stati usati diversi modelli in locale ma anche ChatGPT e Claude AI nelle loro versioni hosted per verificare il comportamento del sistema con modelli più performanti.

3. Prompt Manager

Il processo di prompt engineering è fondamentale per ottenere risposte realistiche dai vari LLM utilizzati. In particolare, si sfruttano le potenzialità di LangChain per l'interazione con il framework Ollama e per la gestione delle conversazioni.

3.1 Prompt Engineering

Tale fase viene suddivisa in due diverse applicazioni:

- **Prompt per la Simulazione del Terminale.** Il primo prompt è stato progettato per far sì che il modello linguistico fornisca risposte identiche a quelle che si otterrebbero da un terminale Linux reale. Il processo ha incluso:
  - Definizione del comportamento atteso
  - Limitazione delle spiegazioni aggiuntive
  - Formato specifico per l'output
  - Istruzioni per evitare messaggi interattivi non richiesti

- **Prompt per l'Assegnazione del Valore F.** Il secondo prompt è stato creato per attribuire un valore "F" ad ogni comando ricevuto. Questo valore classifica i comandi in base al loro impatto sul sistema:
  - F=0: Comandi che leggono o visualizzano informazioni senza apportare modifiche al sistema
  - F=1: Comandi riguardanti le installazioni di pacchetti senza modificare dati esistenti
  - F=2: Comandi che eseguono modifiche a file, cartella o cambiano l'ambiente di lavoro
  - F=3: Comandi che avviano o fermano servizi, scaricano file o elevano i privilegi
  - F=4: Comandi che creano o eliminano file, cartella o eseguono operazioni simili
  - F=5: Comandi che hanno un impatto significativo sul sistema come la modifica di password o la terminazione di processi

Il valore F acquisisce particolare importanza ai fini del suo impiego nell'algoritmo di pruning.

3.2 Algoritmo di Pruning

A causa delle limitazioni intrinseche degli LLM, ovvero una lunghezza di contesto limitata relativa agli input, è stato necessario, per non eccedere tale limite, implementare un algoritmo di pruning il cui funzionamento è il seguente:

1. Tokenizzazione: i comandi vengono trasformati in token tramite la libreria Transformers (AutoTokenizer) di Hugging Face.

2. Definizione della soglia: viene identificato il numero massimo di token supportato dal singolo LLM in uso.

3. Ordinamento della cronologia: la cronologia dei comandi viene ordinata in base al valore F (a partire dal comando con il valore più basso) e poi per timestamp (a partire dal comando meno recente).

4. Calcolo dello score: ad ogni comando viene assegnato un punteggio basato sul suo valore F e sul tempo trascorso dalla sua esecuzione, utilizzando una formula che include un fattore di decadimento esponenziale.

5. Selezione del comando da rimuovere: il sistema identifica il comando con il punteggio più basso. Il punteggio risultante è calcolato sulla base di due elementi:

- Il valore F più basso
- Il tempo trascorso dall'esecuzione del comando

6. Rimozione: il comando selezionato viene rimosso dalla memoria del modello insieme alla sua risposta.

Questo processo viene ripetuto ogni volta che la lunghezza del contesto supera la soglia massima definita, garantendo che i comandi più importanti e recenti vengano mantenuti in memoria dal modello, per migliorare la capacità di risposta di quest'ultimo nel caso di un'interazione prolungata con l'attaccante.

3.3 Gestione della Storia e del Contesto

Il sistema gestisce due tipi di dati:

- **Lista di comandi con metadati:** contiene ogni comando eseguito con il relativo valore di F e il proprio timestamp. Questi dati vengono salvati in un file JSON e sono necessari per poter utilizzare l'algoritmo di pruning ma non vengono aggiunti alla memoria del modello.

- **Cronologia completa dei messaggi:** contiene la sequenza completa di input e output nel formato appropriato per il modello linguistico. Questa sequenza di coppie (query/answer) viene aggiunta alla memoria del modello dopo ogni comando ricevuto. Inoltre, questa cronologia viene salvata in un altro file JSON.

Il processo di aggiornamento della memoria del modello avviene nel seguente modo:

- Il contesto iniziale viene aggiunto alla memoria del modello all'avvio del sistema.
- Successivamente vengono aggiunti ogni comando e la relativa risposta.
- Quando la lunghezza del contesto supera la soglia prefissata, l'algoritmo di pruning rimuove i comandi meno rilevanti.

4. Gestione delle Classi di Inconsistenza

Durante lo sviluppo sono state identificate diverse classi di inconsistenza a partire dalle risposte degli LLM, che rappresentano e raggruppano le problematiche più comuni riscontrate nell'utilizzo degli LLM in locale.

- **Incomprensione del contesto (classe C1):** Il modello non tiene conto delle specifiche fornite nel prompt. Il caso più esplicativo è quando nel prompt viene specificato di non spiegare l'output ma il modello non rispetta questa richiesta.

- **Perdita del contesto (classe C2):** Il modello non mantiene memoria delle interazioni passate. Per esempio, dopo aver creato correttamente una cartella, il modello non ricorda tale operazione e non mostra la cartella precedentemente creata tra quelle disponibili.

- **Incomprensione della query (classe C3):** Il modello non comprende correttamente il comando ricevuto. In particolare, la risposta di un LLM dopo aver ricevuto un comando è identica al comando.

- **Avalanche effect:** Piccoli cambiamenti nel prompt provocano output significativamente diversi.

Per risolvere queste problematiche, alcune funzionalità sono state implementate staticamente nel codice invece di essere delegate completamente al modello:

- **Login tramite SSH:** controllo della validità dell'indirizzo IP e gestione della procedura di autenticazione.
- **Messaggi di benvenuto e uscita:** generazione di messaggi realistici all'inizio e alla fine della sessione SSH.
- **Gestione delle password:** simulazione dell'inserimento delle password per SSH e comandi sudo.
- **Prompt della shell:** corretta rappresentazione dell' utente, dell' hostname del server e del percorso corrente.
- **Gestione dei comandi sudo:** cambio dello username in root, gestione del percorso e del time-out di sessione.
- **Comandi di navigazione (cd):** gestione corretta del cambio di directory e aggiornamento del percorso.
- **Pulizia dell'output:** rimozione di formattazioni non tipiche di un terminale Linux reale.

5. Risultati e Valutazione

Il sistema è stato testato con diversi LLM in esecuzione localmente sfruttando il framework Ollama:

- Mistral:7b
- Gemma2:9b
- Phi4:14b
- Llama3.1:8b
- Qwen2.5:14b

I test hanno evidenziato come i vari modelli forniscano risposte qualitativamente diverse e quali sono le loro capacità legate alla comprensione del contesto e alla memoria.

Il sistema è stato migliorato attraverso un processo iterativo:

1. Implementazione di un prompt di partenza e valutazione dei risultati su diversi LLM.
2. Miglioramento del prompt, evitando esempi in quanto la loro presenza mostra effetti controproducenti e raffinando le istruzioni già presenti nella versione precedente.
3. Integrazione di funzionalità gestite staticamente per risolvere le problematiche più comuni.
4. Adottando questo approccio, come si può notare dalla tabella di seguito, il numero di occorrenze delle varie classi di inconsistenza cala drasticamente.

	Mistral:7b	Gemma2:9b	Qwen2.5:14b	Phi4:14b	Llama3.1:8b
Totale occorrenze C1	5	3	5	2	5
Totale occorrenze C2	14	10	5	9	10
Totale occorrenze C3	30	30	30	30	30

	Mistral:7b	Gemma2:9b	Qwen2.5:14b	Phi4:14b	Llama3.1:8b
Totale occorrenze C1	3	2	3	1	3
Totale occorrenze C2	10	5	5	6	6
Totale occorrenze C3	30	30	30	30	30

	Mistral:7b	Gemma2:9b	Qwen2.5:14b	Phi4:14b	Llama3.1:8b
Totale occorrenze C1	1	2	3	2	2
Totale occorrenze C2	7	2	2	2	3
Totale occorrenze C3	7	7	6	7	10

Inoltre, sono stati effettuati test anche con versioni hosted di ChatGPT e Claude AI, che hanno mostrato un comportamento significativamente migliore grazie alla maggiore capacità di elaborazione e comprensione di questi modelli più avanzati.

6. Conclusioni e Sviluppi Futuri

La combinazione di prompt engineering avanzato, algoritmi di gestione della memoria e implementazione statica di alcune funzionalità critiche ha permesso di superare molte delle limitazioni intrinseche degli LLM, creando un sistema che può efficacemente ingannare e tracciare potenziali attaccanti.

Alcuni sviluppi futuri del sistema sono i seguenti:

- **Supporto per protocolli aggiuntivi:** estensione oltre SSH per includere altri protocolli comuni (HTTP, FTP, ecc.).
- **Personalizzazione dinamica:** possibilità di configurare rapidamente diverse "personalità" di sistema per simulare varie distribuzioni Linux o altri sistemi operativi.
- **Miglioramento dell'algoritmo di pruning:** raffinamento delle tecniche di selezione dei comandi da mantenere in memoria.
- **Utilizzo di modelli più recenti:** test e integrazione con i nuovi modelli linguistici più performanti.

7. Bibliografia e Riferimenti

1. Transformers library – Hugging Face: https://huggingface.co/docs/transformers/index

2. LangChain: https://python.langchain.com/docs/get_started/introduction

3. Ollama: https://ollama.com/

4. Mistral AI: https://ollama.com/library/mistral

5. Meta Llama 3: https://ollama.com/library/llama3.1

6. Qwen: https://qwenlm.github.io/

7. Microsoft Phi-4: https://ollama.com/library/phi4

8. Google Gemma 2: https://ollama.com/library/gemma2

9. Wang, Ziyang, et al. "HoneyGPT: breaking the trilemma in terminal honeypots with large language model." *arXiv preprint arXiv:2406.01882* (2024).

tinexta
defence

Defence Tech | Next |
Donexit | Foramil | Innodesi

Via Giacomo Peroni, 452 – 00131 Roma
tel. 06.45752720 – info@defencetech.it
www.tinextadefence.it

#TinextaDefenceBusiness