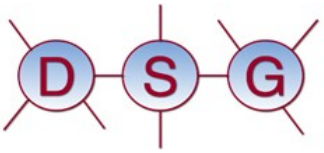

Proposta di Progetto

TITOLO:

Specification-guided vulnerability detection

DESCRIZIONE

Contesto

La vulnerability detection basata su ML/LLM si è evoluta da modelli che trattavano il codice come sequenza di token a metodi che usano strutture come AST, CFG e CPG, fino ad arrivare a pipeline LLM-based con retrieval e ragionamento guidato. Tuttavia, anche i lavori recenti più promettenti restano limitati da diversi problemi strutturali, tra cui la qualità dei dati e la difficoltà di rappresentare la semantica reale della vulnerabilità. In particolare, sono costruiti a partire da commit di patch, sulla base di assunzioni spesso troppo forti, e includono snippet scorrettamente etichettati. Di conseguenza, un modello addestrato direttamente su questi dati rischia di imparare correlazioni superficiali e di produrre falsi positivi o spiegazioni poco affidabili.

Progetto

Questo progetto propone la realizzazione di un sistema di vulnerability detection basato su LLM che produca decisioni verificabili. L'idea centrale è superare la classificazione pura "vulnerabile/non vulnerabile" e adottare un paradigma **specification-guided**: il sistema non deve limitarsi a riconoscere pattern statistici, ma deve verificare se un frammento di codice viola una proprietà di sicurezza ricostruita a partire da fix reali. In particolare, ogni output del sistema dovrà essere collegato a:

- una regola di sicurezza esplicita;
- una o più patch storiche compatibili con tale regola;
- una localizzazione minima nel codice;
- evidenze a favore e contro la presenza della vulnerabilità;
- un possibile suggerimento di correzione minimale.

Il lavoro prevede la costruzione di una pipeline articolata in quattro componenti principali.

1. Selezione del dominio e delle CWE

Per contenere la complessità, il sistema dovrà concentrarsi su un insieme ristretto di vulnerabilità, selezionate in base a criteri di disponibilità di patch storiche leggibili e documentate, e di possibilità di formulare regole operative precise. Un insieme iniziale plausibile comprende, ad esempio CWE-476 (Null Pointer Dereference), CWE-20 (Improper Input Validation) e CWE-22 (Path Traversal).

2. Costruzione della knowledge base di fix e specifiche

Partendo da dateset pubblici selezionati (es. Devign, PrimeVul), per ciascuna CWE selezionata sarà necessario raccogliere un insieme di patch e fix storici. Da ogni caso saranno estratti diversi elementi, come funzione/file pre-fix e post-fix, diff e messaggio di commit, CWE associata e localizzazione della modifica.

Successivamente, a partire da questi dati sarà necessario estrarre delle informazioni strutturate che sintetizzino la regola di sicurezza implicita in ogni patch. Questa fase è delicata e prevede di individuare per ogni CWE una serie di specifiche informazioni, astraendo una conoscenza semantica di alto livello che espliciti quale proprietà di sicurezza la patch introduce o ripristina, cercando di formalizzare in modo riusabile ciò che una patch ha corretto. Verrà quindi costituito un database di *patch specification*.

Esempio minimale di informazioni per CWE-476 potrebbe essere raccolte: precondizione rilevante (una funzione di allocazione può restituire NULL), azione pericolosa (dereferenziazione del puntatore), guardia attesa (controllo esplicito ptr != NULL o gestione dell'errore), indicatore di fix (il commit aggiunge il controllo nullo o sposta la dereferenziazione dentro un ramo protetto).

3. Analisi del codice target e retrieval della specifica

Dato un nuovo frammento di codice, la pipeline deve prima costruire una vista strutturata del frammento e del suo contesto: signature e chiamate principali, eventuali caller/callee immediati, controlli di validazione presenti, API sensibili o sink rilevanti, eventualmente arricchiti mediante rappresentazioni strutturate basate su AST/CPG.

Qui viene eseguito un retrieval ibrido verso la knowledge base delle patch specification, combinando ad esempio: similarità testuale sul diff e sulla descrizione della regola, embedding semantici del codice, filtri strutturali (per esempio presenza di chiamate a API di allocazione, sink di filesystem, funzioni di parsing), ecc. L'obiettivo è recuperare le specifiche più plausibili per il caso corrente.

4. Verifica guidata della regola

Qui il sistema deve essere progettato per rispondere in modo strutturato (e.g., JSON) ad una sequenza di verifiche vincolate, ad esempio: CWE candidate, precondition found, violation found, evidenze dove si osserva la violazione e evidenze nel contesto che la smentiscono.

Tale risposta e il pezzo del sistema per ottenerla dovrà essere progettata in base a specifiche che emergeranno nelle fasi precedenti. L'obbiettivo è ottenere un detector specification-guided in grado di motivare tecnicamente le proprie osservazioni.

Valutazione del prototipo

La valutazione del sistema dovrà basarsi su tre livelli:

- confronto con una baseline di classificazione diretta su un sottoinsieme di di un dataset;

- analisi qualitativa di casi in cui il sistema produce una violazione ben motivata ma il dataset presenta etichette dubbie;
- valutazione della qualità dell'output spiegabile, considerando la regola richiamata, la localizzazione minima, le evidenze riportate e la coerenza con il fix storico.

Risultato atteso

Il risultato atteso è un prototipo di vulnerability detection capace di analizzare un frammento di codice e produrre una decisione verificabile, ancorata a una regola di sicurezza e supportata da evidenze tecniche.

Il contributo principale del progetto non è soltanto nel migliorare la classificazione, ma nel definire un metodo più robusto e spiegabile per la verifica della vulnerabilità, fondato su patch storiche e su proprietà di sicurezza esplicite.

Tecnologie coinvolte

Il progetto prevede l'utilizzo di strumenti e risorse open source, tra cui:

- **Python, JSON/JSONL** o **SQLite, Docker** ;
- dataset open source (e.g., **PrimeVul** e **Devign)**;
- strumenti di analisi statica e strutturale (e.g., **Joern, CodeQL)**;
- modelli LLM e modelli di embedding per codice.

Reference:

- https://www.usenix.org/system/files/usenixsecurity25-lekssays.pdf
- https://dl.acm.org/doi/epdf/10.1145/3728912
- https://dl.acm.org/doi/epdf/10.1145/3715738
- https://dl.acm.org/doi/epdf/10.1145/3796512
- https://arxiv.org/abs/2505.19395
- https://arxiv.org/abs/2511.04014

Referente universitario:

prof. **Francesco Zanichelli** (francesco.zanichelli@unipr.it)
dott. Ing. **Gabriele Penzotti** (gabriele.penzotti@unipr.it)
dott. Ing. **Fabrizio De Santis** (fabrizio.desantis@unipr.it)

Distributed Systems Group, Dipartimento di Ingegneria e Architettura