Construction de benchmarks robustes pour la détection de similarité de code binaire

Encadrants : Nicolas Baskiotis, Benjamin Maudet ,ISIR, Sorbonne Université
(prenom.nom@sorbonne-universite.fr)

Mots-Clés : Binary Code Similarity Detection, Machine Learning, Large Language Models, Apprentissage de représentation

Nombre d'étudiants : 2

Résumé : L'objectif du projet est de construire un benchmark diversifié et réaliste pour la détection de similarité de code binaire en exploitant des implémentations multi-langages de tâches identiques, puis d'évaluer les approches état de l'art sur ce nouveau benchmark.

Sujet :

La *détection de similarité de code binaire* (Binary Code Similarity Detection - BCSD) est un problème fondamental en sécurité informatique et en analyse de logiciels. Il trouve de nombreuses applications pratiques telles que la détection de vulnérabilités, l'analyse de malwares, la vérification de conformité de licences, ou encore la détection de plagiat de code. Ces dernières années, l'apprentissage automatique et en particulier les Large Language Models (LLMs) ont montré des résultats prometteurs pour l'apprentissage de représentations de code binaire [1, 2, 3].

Cependant, un problème majeur limite l'avancée de ce domaine : l'absence de benchmarks robustes, diversifiés et représentatifs des cas d'usage réels. Les benchmarks existants [1, 2] se concentrent principalement sur la compilation d'un même code source avec différents niveaux d'optimisation ou sur des variations cross-architecture (x86, ARM, MIPS). Si ces scénarios sont pertinents, ils restent éloignés de nombreux cas pratiques où l'on cherche à détecter une similarité *sémantique* entre programmes : deux implémentations différentes d'un même algorithme, des variantes d'une même fonctionnalité, ou des codes inspirés d'une même source mais réécrits dans des langages différents.

L'objectif de ce projet est double. Dans un premier temps, il s'agira de construire un benchmark novateur pour le BCSD en exploitant des plateformes comme Rosetta Code et CodinGame. Ces sites proposent de multiples implémentations dans différents langages de programmation pour une même tâche algorithmique. En compilant ces implémentations en code binaire, on obtient naturellement des paires de binaires sémantiquement similaires (ils résolvent le même problème) mais syntaxiquement différents (implémentations distinctes, langages différents). Ce benchmark permettra d'évaluer la capacité des modèles à capturer la similarité sémantique plutôt que de simples variations syntaxiques dues à la compilation.

Dans un second temps, le projet consistera à implémenter et évaluer les approches état de l'art de la littérature sur ce nouveau benchmark. Plusieurs familles d'approches seront considérées : les modèles basés sur les LLMs [1], les architectures de type Transformer adaptées au code assembleur (k-trans) [4], les approches d'embedding comme Asm2Vec [5], ainsi que des méthodes plus classiques basées sur l'extraction de features manuelles. L'analyse comparative permettra d'identifier les forces et faiblesses de chaque approche face à ce nouveau type de

benchmark plus proche des scénarios réels.

Le projet comprendra les étapes suivantes :

1. Collecte et curation des implémentations depuis Rosetta Code et CodinGame
2. Compilation des codes sources en binaires avec différentes configurations (compilateurs, architectures)
3. Construction d'un dataset annoté avec labels de similarité sémantique
4. Implémentation des baselines (features manuelles, graph-based methods)
5. Adaptation et évaluation des approches état de l'art (LLMs, k-trans, Asm2Vec)
6. Analyse comparative des résultats et identification des limites des approches actuelles

Ce projet permettra aux étudiants de se familiariser avec les problématiques de représentation de code binaire, d'apprentissage profond appliqué à la cybersécurité, et de construction rigoureuse de benchmarks pour l'évaluation de modèles de machine learning.

Références

[1] Pei, K., Guan, J., Jungas, M., Grunwald, D., and Jana, S. (2023). Trex : Learning execution semantics from micro-traces for binary similarity. arXiv preprint arXiv :2012.08680.

[2] Li, X., Qu, Y., and Yin, H. (2021). PalmTree : Learning an Assembly Language Model for Instruction Embedding. In Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS '21).

[3] Massarelli, L., Di Luna, G. A., Petroni, F., Baldoni, R., and Querzoni, L. (2019). Safe : Self-attentive function embeddings for binary similarity. In International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (pp. 309-329).

[4] Xu, X., Liu, C., Feng, Q., Yin, H., Song, L., and Song, D. (2017). Neural Network-based Graph Embedding for Cross-Platform Binary Code Similarity Detection. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (pp. 363-376).

[5] Ding, S. H., Fung, B. C., and Charland, P. (2019). Asm2vec : Boosting static representation robustness for binary clone search against code obfuscation and compiler optimization. In 2019 IEEE Symposium on Security and Privacy (SP) (pp. 472-489).