FASER: Binary Code Similarity Search through the use of Intermediate Representations

By Josh Collyer, Tim Watson and Iain Phillips

The Alan Turing Institute

Loughborough University

Background

What is Binary Code Similarity Search?

- Essentially an information retrieval task
- Query function and a corpus of other functions, is my query function present within this corpus?
- Has been applied to a wide range of different task:
    - Identifying N-days within software (lots of focus on firmware and IoT)
    - Identifying open source libraries within binaries
    - Identifying function re-use within software and malware
    - Identifying prior reverse engineered functions

This has been around for a while though right?

- Solved with NLP and Graphs (with a growing trend of combos)
- Approaches such as Asm2vec[1], SAFE[2] and GEMINI[3] pushed the field forward in 10's
- Newer NLP approaches have leveraged transformers such as jTrans[4], PalmTree[5] and Trex[6]
    - Either BERT or RoBERTa
    - Pre-trained -> finetuned
- Sometimes cross architecture, usually mono across compilers + compiler options

Our Contribution

Proposed Approach

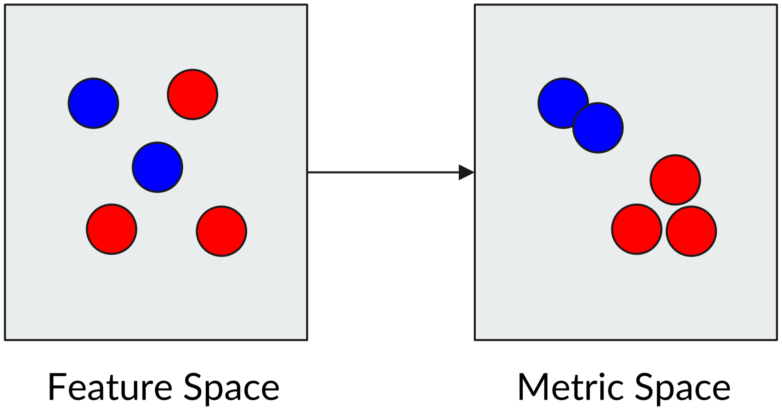
- Use the advancements in long context transformers by leveraging the LongFormer[7] architecture
    - Different attention mechanism = More input tokens
- Ditch the pre-training and train for the objective directly
    - A more targeted, high performing model
- Use an intermediate representation instead of bytes/disassembly
    - Reduced need to normalisation, small vocab size and inherently cross-architecture
- Use deep metric learning, circle loss, online batch hard mining and dynamic pair generation
    - Leverage the research coming out of facial recognition/image retrieval

Proposed Approach

- Use the advancements in long context transformers by leveraging the LongFormer[7] architecture
  - Different attention mechanism = More input tokens
- Ditch the pre-training and train for the objective directly
  - A more targeted, high performing model
- Use an intermediate representation instead of bytes/disassembly
  - Reduced need to normalisation, small vocab size and inherently cross-architecture
- **Use deep metric learning, circle loss, online batch hard mining and dynamic pair generation**
  - **Leverage the research coming out of facial recognition/image retrieval**

The Double Edged Sword - Deep Metric Learning

- Optimisation is driven by a metric - typically distance
- Very frustrating to train
- Usually requires large batch sizes (512+)

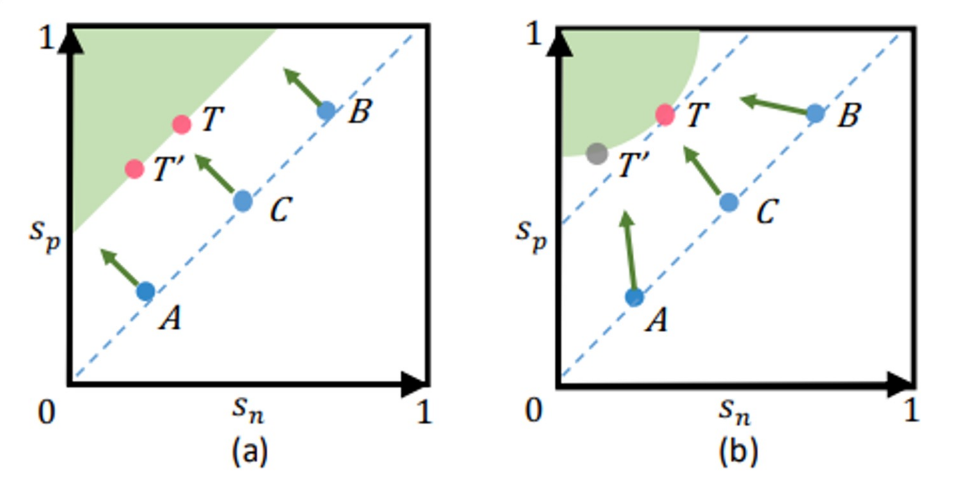


Dynamic Pair Mining

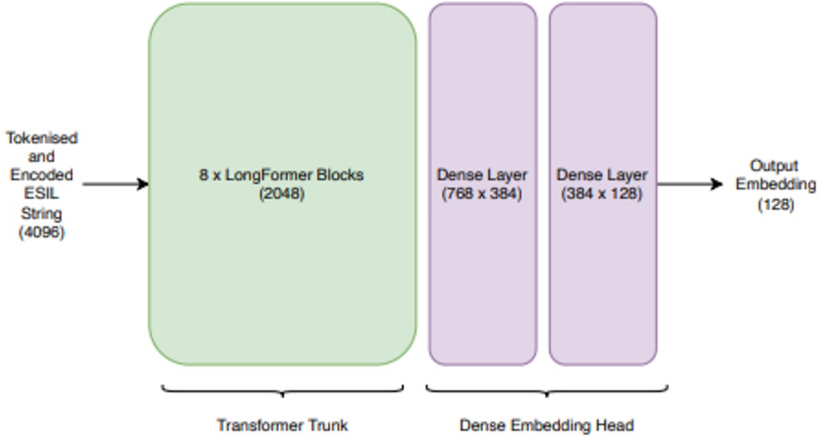
- Most prior research using pre-computed pairs/triplets
- General process is:
  - Embed all examples in batch
  - Dynamically make positive and negatives pairs based on labels
  - Generate losses
  - Take the best/worst/mean/something of the losses and use to update network
- Constantly challenge the models weaknesses

Circle Loss[8]

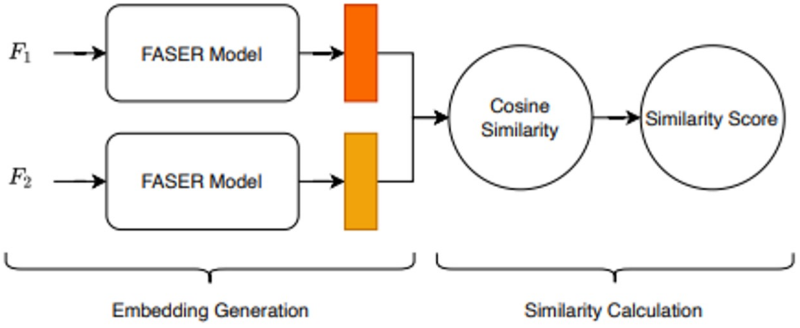
- Used a lot in facial recognition and image retrieval generally
- Uses a circular decision boundary instead of a straight one
- Emphasises suboptimal similarity scores by re-weighting them using a dynamic penalty strength
- Able to deal with large similarity variations at the beginning of training/whilst your learning rate is high



Architecture Diagram



(a) Overview of the Model Architecture Used



(b) Siamese Training Formulation

Dataset Used

- The Dataset-1 and Dataset-Vulnerability from Marcelli et al (2022)[9]
- Dataset-1
    - ClamAV, Curl, nmap, Openssl, Unrar, z3 and zlib compiled using four different version of Clang/GCC over 5 different optimisation levels (1.5M functions once processed)
- Dataset-Vulnerability
    - A 14 OpenSSL CVE's present within the *libcrypto* libraries of two firmware images alongside the same library compiled for arm32, mips32, x86 and x86-64.

Experiments

General Function Search

Objective: Given a query function, can the model correctly retrieve the correct function within a *search pool* of 100 negatives and 1 positive?

- The XM task was used from within Marcelli (2022)
- No constraints on architectures, bitness, compilers or optimisations.
- Hardest and closest to real life

Metrics: Recall@1 (also Precision@1 due to there only being one positive (i.e relevant) function in search pool) & Mean Reripical Rank (MRR)@10 (how far down the ranking the first relevant function is)

Results

		XM	
Method	**Description**	**R@1**	**MRR@10**
FASER NRM	ESIL Function String	**0.51**	**0.57**
FASER RN	ESIL Function String	0.46	0.53
GMN [13]	CFG + BoW opc 200	0.45	0.53
GNN [13]	CFG + BoW opc 200	0.44	0.52
GNN (s2v) [23]	CFG + BoW opc 200	0.26	0.36

Vulnerability Search

Objective: Given the a known vulnerable function within the OpenSSL libcrypto library, can the model identify if a given firmwares OpenSSL libcrypto library also contains the function?

- Approximately ~1000 functions in the firmware OpenSSL library
- Search-pool is effectively 10 times the size

Results reported are for the NETGEAR R700 (arm32) libcrypto vulnerability search.

Results

NETGEAR R700						
	ARM	MIPS	X86	X86-64	Mean Rank	Median Rank
GNN	4:1:1:44	97:5:1:138	3:31:1:18	9:6:1:40	25	5.5
GNN (s2v)	2:1:1:6	35:5:1:7	8:1:5:36	9:1:14:8	8.125	5.5
Trex	32:4:1:2	24:16:1:1	41:4:1:3	10:3:1:2	9.125	3
GMN	1:1:1:1	1:1:1:7	1:1:1:2	1:1:30:7	3.625	1
FASER NRM	1:5:1:1	9:122:1:3	21:21:3:50	7:122:1:2	23.125	4
FASER RN	1:6:1:1	2:4:1:4	2:1:1:3	1:2:1:1	2	1

Zero Shot Experiment

	ARM32	Mean Rank	Median Rank	MIPS32	Mean Rank	Median Rank
FASER NRM	48;546;964;14	393	297	48;30;22;546;170;	154	101.5
		393	297	251;14;155	154	101.5
FASER RM	673;292;1004;15	496	482.5	673;33;4;292;76;	172	106
		496	482.5	136;15;147	172	106

Conclusions

- Forgoing pre-training seems to work
- Using intermediate representations as inputs for function search seems promising
- MIPS still an issue
- Not good enough for zero-shot architecture search
- Work to do on several areas:
  - In-depth understanding of what functions the model struggles with
  - Adoption and development of pre-filtering approaches
  - Integration with other data sources such as decompiled code

MIPS

Anything else

*yet

Questions

References

[1] Ding, S.H., Fung, B.C. and Charland, P., 2019, May. 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). IEEE.

[2] Massarelli, L., Di Luna, G.A., Petroni, F., Baldoni, R. and Querzoni, L., 2019. Safe: Self-attentive function embeddings for binary similarity. In *Detection of Intrusions and Malware, and Vulnerability Assessment: 16th International Conference, DIMVA 2019, Gothenburg, Sweden, June 19–20, 2019, Proceedings 16* (pp. 309-329). Springer International Publishing.

[3] Xu, X., Liu, C., Feng, Q., Yin, H., Song, L. and Song, D., 2017, October. 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).

[4] Wang, H., Qu, W., Katz, G., Zhu, W., Gao, Z., Qiu, H., Zhuge, J. and Zhang, C., 2022, July. Jtrans: Jump-aware transformer for binary code similarity detection. In *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis* (pp. 1-13).

[5] Li, X., Qu, Y. and Yin, H., 2021, November. Palmtree: Learning an assembly language model for instruction embedding. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security* (pp. 3236-3251).

[6] Pei, K., Xuan, Z., Yang, J., Jana, S. and Ray, B., 2020. Trex: Learning execution semantics from micro-traces for binary similarity. *arXiv preprint arXiv:2012.08680*.

References (cont.)

[7] Beltagy, I., Peters, M.E. and Cohan, A., 2020. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*.

[8] Sun, Y., Cheng, C., Zhang, Y., Zhang, C., Zheng, L., Wang, Z. and Wei, Y., 2020. Circle loss: A unified perspective of pair similarity optimization. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 6398-6407).

[9] Marcelli, A., Graziano, M., Ugarte-Pedrero, X., Fratantonio, Y., Mansouri, M. and Balzarotti, D., 2022. How machine learning is solving the binary function similarity problem. In *31st USENIX Security Symposium (USENIX Security 22)* (pp. 2099-2116).