Project Guideline

Objective:

The project could be an original implementation of a new or published idea, widely used techniques/algorithms, or a detailed empirical evaluation of the existing implementation.

Team:

You will work in group of 3 to 5. Unless stated otherwise, you are required to do a demonstration and a presentation (ppt will be used).

Project Breakdown:

<u>Step 1:</u> Please form your group of 3-5 students. Find sample projects in the Appendix. You can also come up with your idea. If you have other ideas which are not included in the Appendix, please talk to the professor about your group project before submitting project proposal.

<u>Step 2:</u> Draft a proposal of your chosen topic (up to 2 pages), including related literatures, tentative method, programming environment you want to use, and tentative schedule etc. Submit it on elearning. Deadline: March. 27, 2019

<u>Step 3:</u> Design, implement and validate your project. Prepare slides for your project.

<u>Step 4:</u> Demonstration: Each group will conduct a 7~10 mins presentation by the end of semester. Slides and program demos are expected in the presentation. The slides should provide proper information, such as motivation for the task, discussion of related literatures, explanations of the details of the approach, performance evaluation, and discussion of results. You should highlight your contributions and conclusions.

Appendix

Several project ideas are listed below. Please read them carefully. If you are interested in any of them, please email me (maryam.imani@utdallas.edu) to share the contact information of mentors with you. In the subject of you email, please include the course title or number.

1. Vulnerability analysis in Smart Contracts

Project description and goals:

The first step is converting EVM bytecode to a series of readable low-level mnemonics[1] annotated with program counter addresses. This conversion is performed by a single linear scan over the bytecode, converting each instruction to its corresponding mnemonic and incrementing a program counter for each instruction and each inline operand. The result of this step is a sequence of program addresses, mnemonics and their arguments. Next, you need to build Control Flow Graph (CFG) from these readable low-level instructions. In CFG, nodes are basic execution blocks, and edges represent execution jumps between the blocks. However, some edges cannot be determined statically at this phase, thus they are constructed on the fly during symbolic execution. Then, you need to extract different features from CFGs and/or the bytecodes. Next, you should apply various classification such as SVM, Random Forest, Neural Network, etc. to categorize the smart contracts to vulnerable or not vulnerable categories. Finally, you need to evaluate the results to complete this project.

Input: Smart Contracts in bytecode

Output: Control Flow Graphs of bytecodes, Features of these graphs, classification evaluation

References:

[Vandal]: https://arxiv.org/abs/1809.03981

[OYENTE]: https://dl.acm.org/citation.cfm?id=2978309

[Node2vec]: https://dl.acm.org/citation.cfm?id=2939754

[GrafeSAGE]:http://papers.nips.cc/paper/6703-inductive-representation-learning-on-large-graphs

2. Large Scale Data Collection and preprocessing in Spark

Project description and goals:

As part of a political event coding pipeline, students are going to design a spark based preprocessing tool that does the following steps

1. Collects data by crawling a set of spanish news websites on a daily basis.
2. Extracts the main content of the article and related metadata (i.e headline, author, date published)

1 In computer assembler (or assembly) language, a mnemonic is an abbreviation for an operation. It is entered in the operation code field of each assembler program instruction.

3. Process the extracted content with and generate universal dependency parse for each sentences within the content (use Apache Spark here)
4. Store the collected data and processed data in a way compatible with event coder's input format in MongoDB
5. Running some deduplication algorithm at content level. (Comparing two articles from different urls and find out whether they cover the same story)

There are several software packages that can be used at different steps. Details will be provided upon request. Key challenges will be design a crawler such a way that can avoid duplicates, implementation of UD-parse generation in streamlined manner, Setting up apache kafka for inter-module communication and content level deduplication.

Input: Set of seed URLs to spanish news sources

Output: Stored data in MongoDB including raw text and processed ones. Structure outline will be provided later.

References:

1. News-please https://github.com/fhamborg/news-please
2. Scrapy https://scrapy.org/
3. Apache Kafka https://kafka.apache.org/
4. SPEC paper https://ieeexplore.ieee.org/document/7474330
5. Universal Dependency https://universaldependencies.org/
6. ufal-udpipe python package.
7. Deduplication

3. Article duplication detection in different languages

Project description and goals: The purpose of this project is detecting similar articles (with similar topics) in different languages. We have a dataset of articles in languages such as English, Arabic and Spanish. Beside, we have a ground truth of the articles which means that which two of them are duplicate. You need to detect these duplication for unseen (test) articles. The idea is extracting hidden topics and using transfer learning to find the similar articles in different languages. You can train the model in for instance En-Sp articles and use it for En-Ar articles.

Input: News articles in different languages

Output: Deduplicated articles in different languages

References:

Zoph, Barret, Deniz Yuret, Jonathan May, and Kevin Knight. "Transfer learning for low-resource neural machine translation." *arXiv preprint arXiv:1604.02201* (2016).

Gong, Hongyu, Tarek Sakakini, Suma Bhat, and Jinjun Xiong. "Document Similarity for Texts of Varying Lengths via Hidden Topics." In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, vol. 1, pp. 2341-2351. 2018.

Mustafa, Ahmad M., Novel Class Detection and Cross-Lingual Duplicate Detection over Online Data Stream, PhD Dissertation, http://libtreasures.utdallas.edu/xmlui/handle/10735.1/5863

4. Detecting exploitable code in C program source code

Project description and goals: The purpose of this project is detecting exploitable source in c code. Given a set of C source code, extract the n-grams and use LSTM deep learning to categorize if it is exploitable or not. The data is present in https://github.com/CGCL-codes/VulDeePecker . The ground truth is all available.

Input: C source files

Output: Categorize to exploitable or not

References

https://arxiv.org/pdf/1801.01681

https://github.com/CGCL-codes/VulDeePecker