2018 3rd International Conference on Information Systems Engineering

A Survey on Source Code Review Using Machine Learning

Wang Xiaomeng, Zhang Tao, Xin Wei
China Information Technology Security Evaluation Center
Beijing, China
e-mail: xiao_meng_wang@163.com

Hou Changyu
China anhua technology Limited company
Beijing, China
e-mail: loger2010@126.com

Abstract—**Source code review constrains software system security sufficiently. Scalability and precision are of importance for the deployment of code review tools. However, traditional tools can only detect some security flaws automatically with high false positive and false negative by tedious reviewing large-scale source code. Various flaws and vulnerabilities show specific characteristic in source code. Machine learning systems founded feature matrixes of source code as input, including variables, functions and files, generating ad-hoc label by distinguish or generation methodologies to review source code automatically and intelligently. Source code, whatever the programming language, is text information in nature. Both secure and vulnerable feature can be curved from source code. Fortunately, a variety of machine learning approaches have been developed to learn and detect flaws and vulnerabilities in intelligent source code security review. Combination of code semantic and syntactic feature contribute to the optimation of false positive and false negative during source code review. In this paper, we give the review of literature related to intelligent source code security review using machine learning methods. It illustrate the primary evidence of approaching ML in source code security review. We believe machine learning and its branches will become out-standing in source code review.**

Keywords-source code review, machine learning, intelligent, flaws and vulnerabilities, feature representation

I. Introduction

This paper represents research results of literature overview of machine learning (ML) in source code review. Code review targets to discovery bugs, security vulnerabilities, and violation of program specifications. Code bugs, weaknesses or vulnerabilities are prevalent in software and can lead to various security problems including deadlock, information leak or system crash, more seriously, some hacker attacks [1]. The WannaCry ransomware attack was a May 2017 worldwide cyberattack by the WannaCry ransomware cryptoworm [2]. The attack began on Friday, 12 May 2017 [3] [4] and within a day was reported to have infected more than 230,000 computers in over 150 countries. [5] [6]Parts of the United Kingdom's National Health Service (NHS) were infected, causing it to run some services on an emergency-only basis during the attack, Spain's Telefónica, FedEx and Deutsche Bahn were hit, along with many other countries and companies worldwide [7]. Some Windows operating systems in China have been infected, and the campus network users have been the first to suffer serious damage. Large numbers of laboratory data and graduation designs have been locked. The application systems and database files of some large enterprises are not able to function properly after being encrypted. These attacks threaten business, governments and consumers, the rate and cost are increasing with annual cost to the global economy from hacker-crime. In 2017, it is estimated that the global security market will reach over 120 billion [1]. Consequently, the earlier vulnerabilities or bugs are detected, the better software system is. As one of the most important part of software security protection, source code review have been served for decades and more intelligent methods have been implanted to mining code vulnerabilities by ML and data mining.

During last decade, ML get to flourish in code review intelligently [8] [9] [10] [11]. State of the art of ML algorithms and applications are described and discussed in this paper. Machine learning has been boomingly applied in source code security tasks, such as code recommendation, source code bug detection and code retrieval. Traditional machine learning system exploits manual feature engineering for code bug detection, which is label motivating and ad-hoc to specific project. As core expansion of machine learning, deep learning methods are capable of capturing complex bottom-up features with little manual operations. Deep learning can intelligently extract code feature of bugs, match program features with that of the queries and predict the probability of the next possible codes [1]. The source code is ultimately text information, mainly related to natural language (NLP) study in deep learning. Unfortunately, considering the structural differences between programming languages and natural languages, existing NLP algorithms are improper for code representation. To extract applicable feature representations, deficiencies of existing schemes and improvement strategies need to be sorted out [12].

In this paper, we give an exhaustive overview of source code review from some aspects below: Section II briefly introduced the classical source code analysis (SCA), including typical verification methods and proceeding flows. Section III illustrated existing machine learning SCA methods, like traditional machine learning approaches and deep learning applications in source code review. Section IV summarized whole the paper to some conclusions, illustrated existing problems and showed some future work and acknowledge the contributors to this paper.

II. Classical Static Code Analysis

Static Code Analysis (SCA) is usually performed as part of a code review and is carried out at the implementation

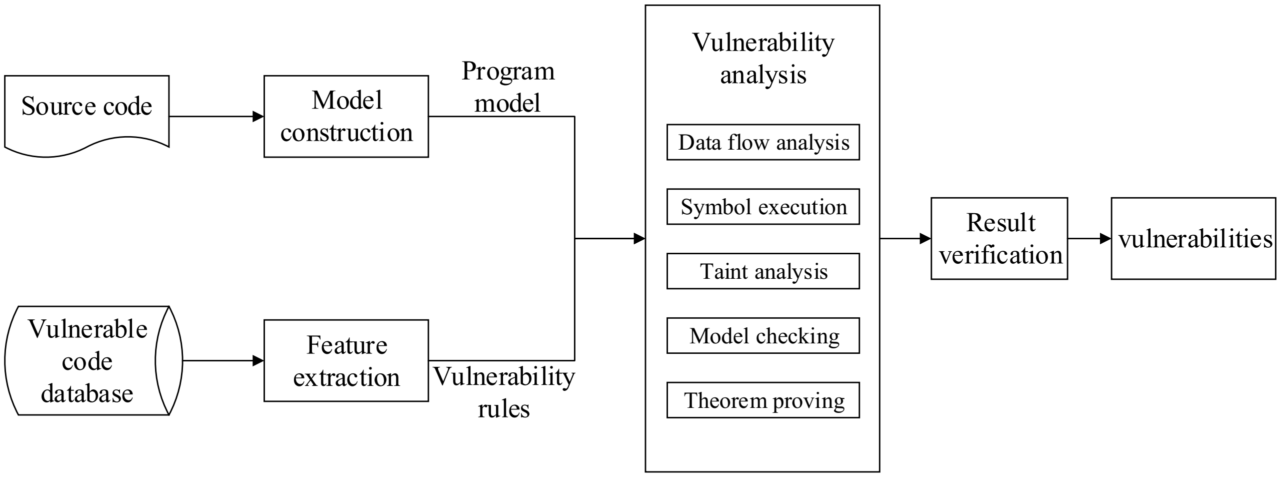

Figure 1. Sketch of classical SCA. Features extracted from vulnerable code unveil the vulnerability rules. Model construction contributes to modeling the program. Classical SCA explores data flow, symbolic execution, taint analysis, model checking and theorem proving to review source code. Review results need to be certificated and categorized. Ultimately, vulnerability collection is established.

phase of a Security Development Lifecycle (SDL). The procedure of SCA. SCA commonly refers to the running of tools that attempt to highlight possible vulnerabilities within 'static' (non-running) source code by using techniques such as Data Flow Analysis, Taint Analysis, and symbolic execution and .etc.

False positive and false negative reveal the limitations of SCA. False positive results might be reported when analyzing an application that interacts with closed source components or external systems because without the source code it is impossible to trace the flow of data in the external system and hence ensure the integrity and security of the data. The use of static code analysis tools can also result in false negative results where vulnerabilities result but the tool does not report them. This might occur if a new vulnerability is discovered in an external component or if the analysis tool has no knowledge of the runtime environment and whether it is configured securely.

Previous works have proven that classical SCA are capable to predict source code vulnerabilities, although sometimes of unacceptable false positive and false negative. Recently, artificial intelligence, especially machine learning and its relative branches have critically influenced academic dialogue on SCA, which would be investigated in the next part.

III. Machine Learning SCA

During last decade, ML get to flourish in intelligent code review on basis of feature representation. ML tasks can be categorized into traditional machine learning, deep learning and reinforcement learning showed in Fig.2. Traditional machine learning is consist of supervised learning and unsupervised learning. For supervised learning, the computer is presented with example inputs and their desired labels, and the goal is to learn a general rule or function that maps inputs to outputs. However, for unsupervised learning, no labels are given to the learning algorithm, leaving it on its own to find structure in its input. Unsupervised learning can be a feature learning procedure.

Deep learning uses a cascade of multiple layers of nonlinear processing units for feature extraction and transformation [13]. Each successive layer uses the output from the previous layer as input, learn in supervised and/or unsupervised manners, learn multiple levels of representations that correspond to different levels of abstraction, and exploits some form of gradient descent for training via back propagation [14]. They may also include latent variables organized layer-wise in deep generative models such as the nodes in Deep Belief Networks and Deep Boltzmann Machines. By contrast, reinforcement learning seems more intelligent that a computer program interacts with a dynamic environment in which it must perform a certain goal. The program is provided feedback in terms of rewards and punishments as it navigates in the problem space. These inputs come from online or offline source code archives.

A. Traditional Machine Learning for SCA

There are various source code archives on the Internet. These archives are usually comprised of application categories and kinds of programming languages. However, manually organizing source code repositories is not a trivial task since they grow rapidly to very large scale. Machine learning methods for automatic classification of archived source code were demonstrated into eleven application topics and ten programming languages. For topical classification, C and C++ programs from the Ibiblio and the Sourceforge archives have dominated research in recent years. Support vector machine (SVM) classifiers are trained on examples of a given programming language or programs in a specified category. Source code can be accurately and automatically classified into topical categories and can be identified to be in a specific programming language. The basic assumptions of the cluster hypothesis and information theory are employed to discover semantically coherent topics in software systems [15]. The usefulness of generated topics is empirically validated

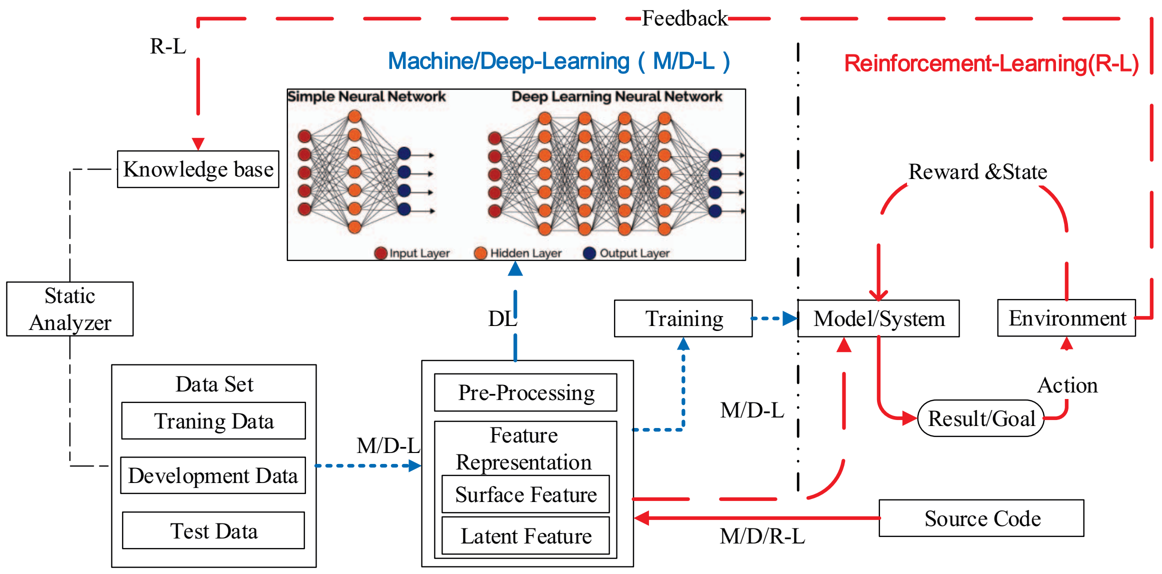

Figure 2. Source code dominate the input in review projects for ML, DL and RL. Data set and knowledge base govern the data space of static analysis in ML and DL. Though pre-processing and feature representation surface and latent feature were extracted as input of training in ML and DL. DL employed a complex multi-layers structure like deep neural network which is different from that in ML, exploiting a simple neural network. Both ML and DL produce a model to predict the input source code slice to which kingdom it belongs. RL raise some distinction. Source code is assembled in preprocessing and feature representation. The results are exported to the RL closed-loop to achieve the Goal by reward and state on the condition of the action on environment. Environment gives feedback to the knowledge base until achieving the goal.

using human judgment. Furthermore, a case study demonstrates that operation of the proposed approach in analyzing code evolution is reported. This proposed approach produces stable, more interpretable, and more expressive topics than classical topic modeling techniques without the necessity for extensive parameter calibration. Methodologies for OS scale program evaluation within a limited time budget are still missing. [3] VDiscovery, a lightweight static and dynamic features were collected to predict if a test case is likely to contain a software vulnerability upon ML. In static level, it could be a good idea to compare similarity between program slices. The smart phone users face a security dilemma: many apps they install operate on privacy-sensitive data, although they might originate from developers whose trustworthiness is hard to judge. Researchers have addressed the problem with more and more sophisticated static and dynamic analysis tools as an aid to assess how apps use private user data [16]. The proposed SUSI, a novel machine-learning guided approach for identifying sources and sinks directly from the code of any Android API. Given a training set of hand-annotated sources and sinks, SUSI identifies other sources and sinks in the entire API. To provide more fine-grained information, SUSI further categorizes the sources and sinks [17]. Many vulnerabilities result from insufficient input validation and thus omitted or missing check provide valuable clues in finding security flaws. Chucky is able to accurately identify missing checks in real time, which ultimately enables us to uncover 12 previously unknown vulnerabilities in two of the projects [18]. Machine learning systems are unique due to either algorithm-intensive nature or applications to potentially large scale data, thus deserving a special consideration. An empirical study is performed to analyze bugs in machine learning system and report the relationship between bug categories and bug severities [19].

B. *Deep Learrning for SCA*

Recently, deep learning has dramatically shaped queries on SCA. Convolutional Neural Networks (CNNs) have gained popularity for handling various NLP tasks. For text classification in particular, deep learning models have achieved remarkable results [1][16][18][20]. The proposed model uses a word embedding layer followed by a convolutional layer with multiple filters, a max-pooling layer and finally a softmax layer. A non-static and randomly initialized embedding layer was employed, so the vectors were trained from scratch. Most common ML task rely on manual feature designing that represent the characteristics of the code. Unfortunately, all of them suffer from challenges in retrieving both semantic and syntactic explanation of source code, an important capability for building accurate prediction models [1]. Convolutional Network is capable for Malware Classification and Analysis. A neural network is implemented that consists of convolutional and feed forward neural constructs. This architecture embodies a hierarchical feature extraction approach that combines convolution of n-grams of instructions with plain vectorization of features derived from the headers of the Portable Executable (PE) files. The evaluation results demonstrate that our approach outperforms baseline methods, such as simple Feed forward Neural Networks and Support Vector Machines, as it achieve 93% on precision and recall, even in case of obfuscations in the data [20].

An emerging approach is treating software code as a form of text and leveraging (NLP) techniques to automatically extract features. Previous work has used BOW to represent a source code file as a collection of code tokens associated with frequencies. The terms are the features which are used as the predictors for their vulnerability pre-diction model.

Classical NLP techniques combined with deep learning are used for non-NLP applications of detection, classification, and reporting of weaknesses related to vulnerabilities or bad coding practices found in artificial constrained languages, such as programming languages and their compiled counterparts [21]. Compare and contrast the NLP approach to the signal processing approach in our results. It shows promising results for specific test cases of open-source software written in C, C++, and JAVA.

Source code is actually text information in nature. The recent advances of deep learning models in machine learning offer a powerful alternative to software metrics and BOW in representing software code. One of the most widely-used deep learning models is Long Short-Term Memory (LSTM) [22], a special kind of recurrent neural networks that is highly effective in learning long term dependencies in sequential data such as text and speech. LSTMs have demonstrated ground-breaking performance in many applications such as machine translation, video analysis, and speed recognition [1]. Some researchers presented a novel deep learning-based approach to automatically learn features for predicting vulnerabilities in software code, leveraging LSTM to capture the long context relationships in source code where dependent code elements are scattered far apart. Powerful deep long short term memory (LSTM) model was established to automatically learn both semantic and syntactic features of code synchronously. Meanwhile, RNN Encoder-Decoder is used for generating API usage sequences for a given API-related natural language query [23]. It may benet other software engineering problems like code search and bug localization. Code clone detection is also an important problem for software maintenance and SCA [24]. The proposed code analysis tool took ad-vantage of deep learning and automatically linked patterns mined at the lexical level with patterns mined at syntactic level.

IV. Conclusion and Future Work

This paper represents research results of literature overview of machine learning (ML) in source code review. ML and some branches raised some new idea to implement classifiers, regression and agent to predict source code vulnerability with lower false positive and false negative. Fortunately, the ability for deep processing of security networks to automatically adjust and tune connections with increasing volumes of data will improve the process of learning. In particular, this will allow us to automate and use networks to specialize in certain areas. With deep learning, the security system can automatically learn by trying billions of combinations and making millions of observations. It is very promising when aimed at a specific class of problems, but it is not a silver bullet. Just because a technology uses deep learning doesn't mean other traditional AI and machine learning approaches are not more valuable or practical. Artificial intelligence is multi-purpose technology we can put to work in security and other industries as well, learning, iterating, and improving as we go. Unfortunately, no paper was found to review source code by Reinforcement learning. The main reason is that during the review no dynamic interaction was established, however, it proposed promising way to combine static and dynamic way to measure source code.

Above all, it still a challenge that vulnerable data is not enough in scale. Cross-file, cross-version and cross-projects need to be dominated, because the object mostly is not exist in the training set. Review by ML, DL and RL has propelled to the forefront in investigations of source code analysis.

Acknowledgment

This paper was supported by National Natural Science Foundation of China (NSFC) U1636115 and NSFC U1736209.

References

[1] Hoa Khanh Dam, Truyen Trany, Trang Pham. "Automatic feature learning for vulnerability prediction". ArXiv: 1708.02368v1 [cs.SE]. 2017

[2] Cameron, Dell. "Today's Massive Ransomware Attack Was Mostly Preventable; Here's How To Avoid It". Gizmodo. 2017.

[3] Reuters. "Shadow Brokers threaten to release Windows 10 hacking tools". The Express Tribune. 2017.

[4] Brenner, Bill. "WannaCry: the ransomware worm that didn't arrive on a phishing hook". Naked Security. Sophos. 2017.

[5] "Cyber-attack: Europol says it was unprecedented in scale". BBC News. 13 May 2017. Retrieved 13 May 2017.

[6] "Unprecedented' cyberattack hits 200,000 in at least 150 countries, and the threat is escalating". CNBC. 14 May 2017. Retrieved 16 May 2017.

[7] Larson, Selena (12 May 2017). "Massive ransomware attack hits 99 countries". CNN. Retrieved 12 May 2017.

[8] Kim, Y. (2014). Convolutional Neural Networks for Sentence Classification. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP 2014), 1746–1751. 2014

[9] Wang, P., Xu, J., Xu, B., Liu, C., Zhang, H., Wang, F., & Hao, H. Semantic Clustering and Convolutional Neural Network for Short Text Categorization. Proceedings ACL 2015, 352–357. 2015

[10] Zhang, X., Zhao, J., & LeCun, Y. "Character-level Convolutional Networks for Text Classification". 1–9. 2015

[11] Mokhov, Serguei A. "The use of machine learning with signal- and NLP processing of source code". Technical Report. NIST. 2011.

[12] Bogdan Dit, Meghan Revelle, Malcom Gethers. "Feature Location in Source Code". Journal of Software Maintenance and Evolution. 2011

[13] Deng, L.; Yu, D. "Deep Learning: Methods and Applications". Foundations and Trends in Signal Processing. 7 (3–4): 1–199. 2014

[14] Bengio, Yoshua. "Learning Deep Architectures for AI". Foundations and Trends in Machine Learning. 2 (1): 1–127. 2009

[15] Anas Mahmoud, Gary Bradshaw. "Semantic topic models for source code analysis". Empirical Software Engineering archive, Volume 22 Issue 4, August 2017, Pages 1965-2000

[16] Gustavo Grieco, Guillermo Luis Grinblat, Lucas Uzal. "Toward large-scale vulnerability discovery using Machine Learning". Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy. Pages 85-96

[17] Siegfried Rasthofer, Steven Arzt, Eric Bodden. "A Machine-learning Approach for Classifying and Categorizing Android Sources and Sinks". NDSS'14, 2014

[18] Fabian Yamaguchi, Christian Wressnegger, Hugo Gascon. “Chucky: Exposing Missing Checks in Source Code for Vulnerability Discovery”. CCS’13, November 4–8, 2013

[19] Ferdian Thung, Shaowei Wang, David Lo. “An Empirical Study of Bugs in Machine Learning Systems”. 2012 IEEE 23rd International Symposium on Software Reliability Engineering (ISSRE).

[20] Bojan Kolosnjaji, Ghadir Eraisha, George Webster. “Empowering Convolutional Networks for Malware Classification and Analysis” 2017 International Joint Conference on Neural Networks (IJCNN),pp 3838-3845

[21] Serguei A. Mokhov, Joey Paquet, and Mourad Debbabi. “The Use of NLP Techniques in Static Code Analysis to Detect Weaknesses and Vulnerabilities”. AI 2014: Advances in Artificial Intelligence pp 326-332

[22] Sepp Hochreiter; Jürgen Schmidhuber (1997). "Long short-term memory". Neural Computation. 9 (8): 1735–1780. doi:10.1162/neco.1997.9.8.1735. PMID 9377276.

[23] Xiaodong Gux, Hongyu Zhangy, Dongmei Zhang. “Deep API Learning”. FSE’16, Nov 13–18, 2016

[24] Martin White, Michele Tufano, Christopher Vendome. “Deep Learning Code Fragments for Code Clone Detection”. ASE’16, September 03-07, 2016,