ICASSP 2024 - 2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) | 979-8-3503-4485-1/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICASSP48485.2024.10447552

CPMSVD: CROSS-PROJECT MULTICLASS SOFTWARE VULNERABILITY DETECTION VIA FUSED DEEP FEATURE AND DOMAIN ADAPTATION

Gewangzi Du, Liwei Chen, Tongshuai Wu, Chenguang Zhu, Gang Shi

Institute of Information Engineering, Chinese Academy of Sciences
Beijing, China

ABSTRACT

Many deep learning-based approaches have achieved excellent performance for Software Vulnerability Detection(SVD) but the most imperative issue is coping with the scarcity of labeled software vulnerabilities. When employing transfer learning techniques, researchers only detected the presence of vulnerabilities but cannot identify vulnerability types. In this paper, we propose the first system for Cross-Project Multiclass Software Vulnerability Detection (CPMSVD) which incorporates inter-procedure code lines as local feature and detects at the granularity of code snippet. Principles are defined to generate snippet attentions and a deep model is proposed to obtain the fusion representations. We then extend domain adaptation techniques to reduce feature distributions among different projects. Experimental results show that our approach outperforms other state-of-the-art ones.

Index Terms— cyber security, deep learning, feature fusion, domain adaptation, multiclass detection

1. INTRODUCTION

Software vulnerabilities dreadfully undermine the security of computer systems due to their ubiquity and the number of vulnerabilities is increasing drastically each year according to Common Vulnerabilities and Exposures (CVE) [1] and National Vulnerability Database (NVD) [2]. Traditional static SVD approaches use manually-defined patterns to detect vulnerabilities [3, 4, 5] but failed to achieve decent performance owning to the incomplete vulnerability patterns giving by experts. Dynamic analysis such as fuzzing [6, 7, 8] and taint analysis [9] inspect vulnerabilities during the program execution but have low code coverage.

With the rapid development of deep leaning technology, researchers leverage deep neural networks to relive human experts from the arduous task of manually defining patterns. [10] proposed VulDeePecker, which is the first SVD system based on deep learning and extracted code slices from API calls through dataflow propagations. An in-domain multiclass vulnerability detection system was proposed in [11] by combining different kinds of features. [12] proposed SySeVR, which is a binary classification system and expands the slicing points. [13] embedded features from code property graph and applied attention mechanism to capture potential vulnerable code lines. [14] detected vulnerabilities using a pre-trained BERT. However, all existed approaches split training and test data from a same dataset, which means deep model built by the training data is able to directly apply to the test data.

A severe challenge with deep learning is the scarcity of labeled projects. Models trained from labeled projects cannot be generalized to a new project because of the distribution discrepancy. Several researches adopted transfer learning skills to solve this problem. [15] employed BiLSTM to learn the transferable representation between projects and used a random forest as downstream classifier. Research in [16] combined the heterogeneous data of code text and abstract syntax tree (AST) to learn unified representations of vulnerability patterns. [17] employed adversarial learning framework to learn domain-invariant features that can be transferred from source to target project. The most recent cross-project SVD research [18] adopted RNN to learn high-level features and utilised a metric transfer learning framework to transform these features. However, all these cross-project researches cannot figure out the vulnerability types. Moreover, they detect vulnerabilities at function level, which is difficult to locate vulnerable lines and not able to capture inter-procedure vulnerability patterns caused by function calls.

We propose our CPMSVD system and the contributions are summarized as follows: Firstly, CPMSVD is a novel system which dedicates to detecting cross-project inter-procedure multiclass vulnerabilities. We carried out experiments on real-world open source projects to verify the effectiveness of our system. Secondly, inspired by fine-grained image recognition in computer vision [19, 20], we defined several types of code statement and several principles to generate snippet attentions. We proposed a deep feature fusion structure, which combines the deep features of two models that accommodate code snippets/snippet attentions to obtain the fused deep features. Finally, we extend different domain adaptation techniques at different specific stages of our system to draw close probability distributions of different projects. We devise to align multiple source projects before involving the target projects.

2. METHODOLOGY

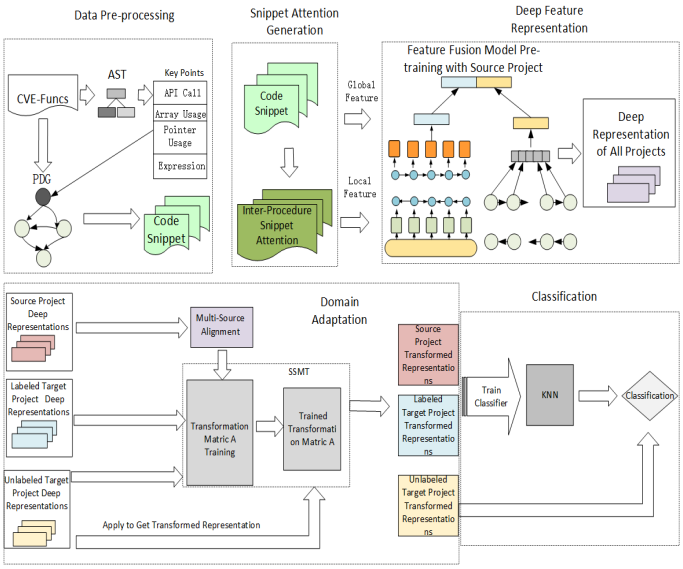

Fig. 1: Overview of CPMSVD.

2.1. Data Pre-processing

The structure of system is shown in Fig.1 and we detect at the granularity of code snippet, which refers to the SeVC in [12]. Code snippets contain significantly less line numbers than functions and span across functions to capture inter-procedure vulnerabilities. We follow the same procedures in [12] to obtain labeled code snippet of real-word projects.

2.2. Snippet Attention Generation

Inspired by region attention [20], we focus on several types of statements in code snippets, which provide strong basis for our cross-project multiclass detection. For example, pointer usage gives information on pointer-related vulnerabilities such as null pointer dereference while array usage easily leads to buffer-related vulnerabilities. APIs such as *memset()* can cause buffer overflows. Variable definition statements related to API/Library calls and pointer/array usage are also important since they may provide information to identify improper use of API/Library and pointer/array. Furthermore, we take into consideration control statements because they may conduct proper bounds-checking and security screening. Thus, we define variable definition statements as "definition" type, pointer/array usage statements as "usage" type, API/Library call statements as "API" type, control statements such as "if" or "while" as "control" type. We propose the following principles to get the inter-procedure snippet attentions:

1) Match between "definition" statements and "usage" statements: If there is a variable defined in a "definition" statement that matches a pointer/array variable in use, which is denoted by symbols such as "$*$","[]" and "$\rightarrow$" in a "usage" statement, we select both the "definition" statement and "usage" statement as snippet attention.

2) Match between "definition" statements and "API" statements: If there is a variable defined in a "definition" statement that matches an argument in a "API" statement, we select both the "definition" statement and "API" statement. We do not choose "API"s without variables in "definition" statements as its argument.

3) Since the parenthesized expressions after "control" statements keywords such as "for" may contain variable definitions, we extract all "control" statements as snippet attention unless there is no variable in the expressions.

4) Inter-Procedure principle: We replace the formal parameters in a callee with their corresponding actual arguments in the calling functions thus the first three principles work across function calls. For nested function calls, we make this parameter transformations inward from outside.

We implement an automate lexical-analysis-based program to match the principles on statements from scratch. We set a threshold τ which is 300. For snippet attentions containing fewer words than τ, we pad zeros at the tail to make their vectors of length τ when vectorize them using the word2vec tool with Continuous Bag-of-Words. For snippet attentions containing more words than τ, we cut them to τ by reserving the key data, which includes the parameters of the conditional branch, the parameters of Library/API call, function pointers, pointers plus offset and pointers assigned by a calculated value. We give a higher priority to the key data and keep them not removed from snippet attentions, then cutting the words of non-key data at the tail to match τ. If the sequences of key data are longer than τ, we cut at the tail of key data.

2.3. Deep Feature Representation

Our network architecture, as shown in Fig.2, is composed of two sub-models. The ASTNN [21], which can capture syntactic information, is reimplemented as global feature model to learn deep features from code snippets. The BiGRU is employed to learn deep local features from snippet attentions. We generate tokens as corpus from code snippets/snippet attentions, discarding non-ASCII characters and comments. We use our lexical-analysis-based program to parse each statement in code snippets respectively and get the ASTs of each statements as ST-trees, which are inputs for ASTNN. The statement encoder layer in the ASTNN can remove the duplicated nodes and convert the ST-trees to binary ST-trees (i.e, right sibling as right child) to capture code naturalness of our ST-trees. This layer also incudes the word2vec, which takes the inorder traversal of ST-trees as corpus for training.

Then we use the labeled data of source project to pre-train the two models. To improve the performance, we pre-train the two models separately and adopt the layer-wise pre-training

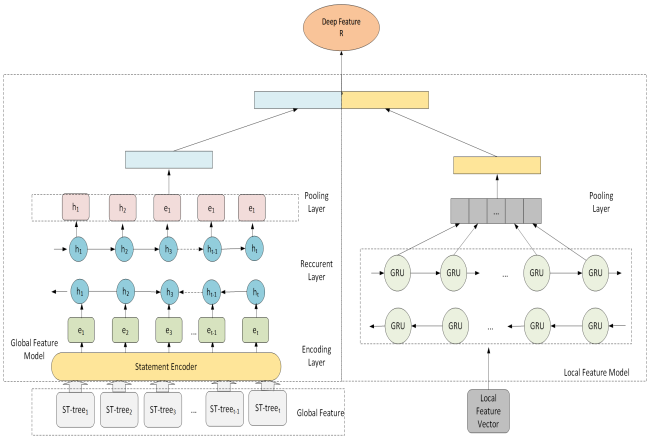

Fig. 2: The neural network architecture.

[22], which is also used in [18]. Finally we feed source and target projects into the pre-trained model to obtain the deep representations for all domains.

2.4. Domain Adaptation

We aim to elminate the distribution discrepancy of deep features among different domains. The CORAL [23] is extended to align all the source projects, which is applied in step 1. In the next step, the SSMT [24] is used to draw close data distribution of the same class between source and target projects.

Step1: Multi-Source Alignment. We align the distribution of different source domains by exploring their covariance, which reflects the distribution of a dataset. We whiten all the source domains, discarding the re-coloring procedure to the target. For each source project S_i, our alignment method is illustrated as:

$$\begin{aligned} C_i &= cov(S_i) + eye(size(S_i, 2)) \\ S_i^* &= S_i * C_i^{\frac{-1}{2}} \end{aligned} \tag{1}$$

Where S_i^* is the adjusted source project. Each adds the correlation of the source domain which has the smallest between group variance and thus all the source domains are aligned.

Step2: Regularized Distance Metric Learning. This step aims to minimize intra-class and maximizing inter-class distance between source and target projects. At the core of this algorithm is SSMT, which reduces the distribution between domains both statistically and geometrically. This algorithm need a handful of labeled data in target domain. SSMT takes data in both source and target domains as input to iteratively update the transformation matrix **A**.

2.5. Classification

Since $(x_i - x_j)\mathbf{A}$ in the objective function [24] is same as $x_i\mathbf{A} - x_j\mathbf{A}$ and $x\mathbf{A}$ can be considered as projecting x into a new space by matrix **A**. In this new space, samples with same labels are get closed. On these transformed features, we put a KNN as the final classifier trained by all labeled data and pinpoint vulnerability types in target domain.

3. EXPERIMENT IMPLEMENTATION

3.1. Experimental Setup

Dataset: We collect code snippets of Linux Kernel (Linux for short), Qemu, Wireshark, Firefox FFmpeg. To validate the robustness of our system, we obtain as many vulnerability types as these projects contain, which lead to 10 different classes. The dataset is shown in Table 1. We combine four of the them as the existing labeled source projects to leverage more training data and the remaining one is target project to be tested, leading to 5 testcases.

Table 1: Dataset and Label

CWE Type(Label)	*Linux*	*Qemu*	*Wireshark*	*Firefox*	*FFmpeg*
Non-vulnerable(0)	560	567	589	522	435
CWE-119(1)	224	80	294	183	202
CWE-399(2)	175	42	217	68	56
CWE-20(3)	137	33	111	29	42
CWE-189(4)	82	28	125	27	79
CWE-835(5)	106	98	102	10	15
CWE-416(6)	103	86	27	16	20
CWE-200(7)	57	96	12	59	15
CWE-787(8)	45	88	14	30	11
CWE-190(9)	43	72	19	23	17

Parameters: We set learning rate 0.01, epoch 60 and batch size 32 for each model. The embedding size of word2vec is 40 and the hidden dimension of BiGRU is 128. We make K equal 1 in KNN. Same as CD-VulD [18], we hold out 30% as labeled data in target domain for training Matrix **A** and the rest is for test. The training and test data are not drawn from overlapped functions.

Evaluation Metric: We include Macro-Averaged False Positive Rate (*M_FPR*), Macro-Averaged False Negative Rate (*M_FNR*), Macro-Averaged F1-measure (*M_F1*), Weighted-Averaged False Positive Rate (*W_FPR*), Weighted-Averaged False Negative Rate(*M_FNR*) and Weighted-Averaged F1-measure(*W_F1*) as evaluation metric [11].

3.2. Experimental Results

1) We perform 10 random permutation to report the mean and standard deviation in Table 2.

For fairness, for SySeVR [12] and Dual [17], we train the model with the same small piece of labeled target data. Pure deep learning system [12] without any transfer learning techniques completely fail in this task. CPMSVD$^-$ represents removing multi-source alignment from our system, and the performance of which is obvious worse than the whole system due. Dual and CD-VulD [18] both adopts domain adaptation

Table 2: Experimental results for cross-project detection (%).

Target	Methods	M_FPR	M_FNR	M_F1	W_FPR	W_FNR	W_F1
Fire fox	SySeVR	12.6±0.3	63.3±1.1	36.2±1.0	29.9±0.5	55.8±0.8	48.5±0.8
	Dual	6.7±0.2	41.3±1.1	55.2±1.0	9.2±0.6	36.5±1.0	72.0±0.8
	CD-VulD	6.2±0.1	34.7±0.9	67.1±0.7	6.9±0.6	28.5±0.8	79.2±0.7
	CPMSVD$^-$	3.4±0.6	23.9±1.4	72.7±1.1	4.8±0.7	17.8±1.2	86.0±0.9
	CPMSVD	**3.0±0.2**	**15.6±0.8**	**78.8±0.4**	**4.7±0.4**	**13.8±0.5**	**88.9±0.5**
Qemu	SySeVR	17.8±0.6	64.3±2.7	40.0±2.8	21.8±1.3	65.7±2.1	40.7±2.0
	Dual	5.3±0.3	33.6±1.9	66.2±1.1	6.8±0.4	28.5±1.8	73.3±0.8
	CD-VulD	4.7±0.3	34.2±1.2	70.6±0.9	7.1±0.3	29.0±0.9	72.4±0.7
	CPMSVD$^-$	3.1±0.2	18.1±1.2	81.5±0.6	6.2±0.4	15.4±0.9	85.5±0.7
	CPMSVD	**2.3±0.2**	**14.4±0.5**	**86.5±0.4**	**5.0±0.3**	**11.1±0.4**	**89.4±0.3**
FF mpeg	SySeVR	20.1±0.3	67.0±4.1	38.2±3.5	10.6±0.9	68.6±2.5	42.2±1.7
	Dual	6.0±0.7	38.9±3.0	61.8±2.5	6.7±0.9	34.2±2.3	72.7±1.0
	CD-VulD	6.5±0.5	37.3±2.6	62.6±1.5	**5.7±0.6**	30.0±2.2	77.4±1.3
	CPMSVD$^-$	4.1±0.2	27.6±0.8	68.3±0.5	6.7±0.2	23.3±0.7	79.9±0.5
	CPMSVD	**3.9±0.3**	**22.7±0.7**	**72.4±0.6**	5.8±0.3	**19.0±0.6**	**84.5±0.4**
Wire shark	SySeVR	10.1±0.7	61.2±3.9	41.6±3.8	11.0±0.6	52.8±3.3	52.3±2.9
	Dual	4.5±0.4	28.6±1.9	70.1±1.0	7.1±0.5	24.7±1.6	77.0±1.2
	CD-VulD	4.7±0.4	31.3±1.2	67.4±0.8	6.2±0.6	25.8±1.1	77.3±0.7
	CPMSVD$^-$	2.3±0.3	12.3±0.7	81.4±0.6	3.7±0.2	12.9±0.6	87.8±0.5
	CPMSVD	**1.6±0.2**	**9.9±0.6**	**86.2±0.5**	**2.4±0.3**	**8.7±0.5**	**92.4±0.4**
Linux	SySeVR	15.6±0.8	50.9±3.9	52.2±2.4	11.1±1.3	56.2±3.2	51.4±2.0
	Dual	5.2±0.5	27.3±1.2	72.9±0.9	5.3±0.7	29.5±1.1	74.9±1.0
	CD-VulD	5.6±0.4	28.2±1.1	73.5±0.9	9.3±0.5	30.6±1.0	73.0±0.8
	CPMSVD$^-$	2.8±0.4	18.4±0.8	81.7±0.5	3.8±0.7	15.2±0.5	85.4±0.5
	CPMSVD	**2.0±0.2**	**14.1±1.0**	**86.3±0.6**	**3.4±0.3**	**11.0±0.9**	**89.4±0.6**

techniques and CD-VulD is slightly better than Dual in general, but both of them are obviously worse than CPMSVD. To summarize the overall evaluation of the 5 testcases, our average *M_F1*, *W_F1* and *M_FNR*, *W_FNR* are respectively 13.8%, 12.9% higher and 17.8%, 16.1% lower than CD-VulD. While compared with Dual, our average *M_F1*, *W_F1* and *M_FNR*, *W_FNR* increase by 17.0%, 13.9% and decrease by 18.6%, 18.1%.

2) To validate the effectiveness of our deep representation methods, we conduct abundant of comparative experiments. We replace our deep feature extraction part with the corresponding part in CD-VulD and rename the it to CD-VulD$^+$. We also remove local feature part from our system and rename it to ASTNN. These two baselines using only code snippet as input. To compare with μvuldeepecker [11], we replace our local feature with their local feature, keeping other parts of our system unchanged, which is renamed to μvuld*. Besides, we replace our reimplemented ASTNN in the system with the original one and rename it to CPMSVD$^\#$. To evaluate our key data preservation method when handling the snippet attention, we simply cut or pad zeros at the end of snippet attentions to match τ and rename it to CPMSVD*. Experimental results are shown in Table 3.

The performance of CD-VulD$^+$ and ASTNN are far behind other two approaches, indicating that local features are important for multiclass tasks. CPMSVD$^\#$ and CPMSVD* are also worse than CPMSVD. Moreover, CPMSVD achieves better results than μvuld* due to our more complete local features which help distinguish more vulnerability patterns

Table 3: Evaluating all deep feature extracting strategies (%).

Target	Methods	M_FPR	M_FNR	M_F1	W_FPR	W_FNR	W_F1
Fire fox	CD-VulD$^+$	4.2±0.2	26.5±1.0	70.1±1.0	7.9±0.6	22.6±1.0	81.3±0.8
	ASTNN	3.7±0.1	25.9±0.9	71.2±0.7	7.5±0.6	22.1±0.8	81.9±0.8
	μvuld*	3.6±0.3	20.5±0.9	73.4±0.8	5.5±0.4	18.3±0.8	85.1±0.6
	CPMSVD$^\#$	4.0±0.2	19.5±0.8	73.0±0.6	5.4±0.5	18.1±0.6	85.5±0.6
	CPMSVD*	3.5±0.2	18.2±0.9	75.9±0.7	5.3±0.4	15.0±0.7	86.7±0.5
	CPMSVD	**3.0±0.2**	**15.6±0.8**	**78.8±0.4**	**4.7±0.4**	**13.8±0.5**	**88.9±0.5**
Qemu	CD-VulD$^+$	3.4±0.3	21.3±1.4	79.0±0.1	5.7±0.4	18.1±1.3	82.1±0.8
	ASTNN	3.1±0.3	20.5±1.2	80.9±0.9	5.3±0.4	17.7±0.9	83.0±0.7
	μvuld*	2.6±0.2	16.7±0.6	83.2±0.4	5.9±0.2	12.9±0.6	87.4±0.5
	CPMSVD$^\#$	**2.2±0.2**	15.8±0.9	83.9±0.7	5.5±0.4	13.2±0.5	87.7±0.5
	CPMSVD*	2.6±0.3	15.6±0.6	84.9±0.5	5.7±0.3	12.2±0.6	88.3±0.5
	CPMSVD	2.3±0.2	**14.4±0.5**	**86.5±0.4**	**5.0±0.3**	**11.1±0.4**	**89.4±0.3**
FF mpeg	CD-VulD$^+$	5.5±0.3	33.6±0.8	61.0±0.6	6.9±0.4	32.1±0.7	71.5±0.5
	ASTNN	5.0±0.2	32.3±0.7	62.5±0.5	6.8±0.4	29.4±0.6	73.8±0.5
	μvuld*	4.2±0.1	27.1±0.9	67.2±0.7	6.5±0.3	24.9±0.8	78.0±0.6
	CPMSVD$^\#$	3.9±0.2	25.6±0.9	68.2±0.6	6.3±0.4	24.4±0.7	79.5±0.6
	CPMSVD*	4.3±0.2	25.2±0.8	68.9±0.5	6.1±0.3	22.2±0.7	80.3±0.5
	CPMSVD	**3.9±0.3**	**22.7±0.7**	**72.4±0.6**	**5.8±0.3**	**19.0±0.6**	**84.5±0.4**
Wire shark	CD-VulD$^+$	3.4±0.3	18.3±1.8	76.6±0.8	4.8±0.4	18.5±1.4	83.1±0.6
	ASTNN	3.2±0.2	16.8±1.9	78.3±1.1	4.6±0.4	17.7±1.6	83.9±0.7
	μvuld*	2.8±0.3	13.4±0.7	81.1±0.6	4.4±0.3	15.1±0.5	86.0±0.5
	CPMSVD$^\#$	2.5±0.3	11.7±0.8	83.0±0.7	4.1±0.5	14.2±0.7	87.9±0.7
	CPMSVD*	2.3±0.3	11.5±0.8	83.6±0.6	4.0±0.4	12.7±0.7	88.1±0.6
	CPMSVD	**1.6±0.2**	**9.9±0.6**	**86.2±0.5**	**2.4±0.3**	**8.7±0.5**	**92.4±0.4**
Linux	CD-VulD$^+$	4.1±0.5	24.6±1.2	76.1±0.9	3.9±0.7	21.9±1.1	80.8±1.0
	ASTNN	3.8±0.4	22.1±1.1	77.9±0.9	3.7±0.4	20.9±1.0	81.5±0.8
	μvuld*	2.8±0.3	18.1±0.7	81.6±0.6	**3.2±0.4**	15.7±0.7	85.6±0.6
	CPMSVD$^\#$	2.7±0.3	18.6±0.6	81.1±0.5	3.6±0.3	17.1±0.5	83.8±0.4
	CPMSVD*	2.5±0.3	16.3±0.7	83.9±0.6	3.5±0.4	13.6±0.7	86.9±0.5
	CPMSVD	**2.0±0.2**	**14.1±1.0**	**86.3±0.6**	3.4±0.3	**11.0±0.9**	**89.4±0.6**

precisely. Compared with μvuld*, our average *M_F1*, *W_F1* and *M_FNR*, *W_FNR* increase by 4.7%, 4.5% and decrease by 3.8%, 4.7%.

We furthermore combine the five projects as source domain to detect the swftools [25] with scarcity of labeled data as target and discover two undisclosed vulnerabilities which are not reported in the NVD using our CPMSVD. We open the issues at https://github.com/matthiaskramm/swftools/issues/210(211) and CVE IDs will be assigned to these zero-day vulnerabilities.

4. CONCLUSION

We firstly designed a system to pinpoint software vulnerability types across different projects with inter-procedure features. We leverage region attention and feature fusion model for high-level feature extracting. Moreover, domain adaptation techniques are used to eliminate distribution divergency between different projects.

5. ACKNOWLEDGMENT

This work is partially supported by the National Natural Science Foundation of China (No. 62172407), and the Youth Innovation Promotion Association CAS.

6. REFERENCES

[1] "CVE," https://cve.mitre.org/.

[2] "NVD," https://nvd.nist.gov/.

[3] "Checkmarx," https://www.checkmarx.com/.

[4] "Coverity," https://scan.coverity.com/.

[5] "CppChecker," http://cppcheck.sourceforge.net/.

[6] M. Sutton, A. Greene, and P. Amini, "Fuzzing: brute force vulnerability discovery,"*Pearson Education,* 2007.

[7] H. Chen, Y. Xue, Y. Li, B. Chen, X. Xie, X. Wu, and Y. Liu, "Hawkeye:Towards a desired directed grey-box fuzzer," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security,* 2018, pp. 2095-2108.

[8] Y. Li, Y. Xue, H. Chen, X. Wu, C. Zhang, X. Xie, H. Wang, and Y. Liu,"Cerebro: context-aware adaptive fuzzing for effective vulnerability detection," *in Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering,* 2019, pp. 533-544.

[9] J. Newsome and D. Song, "Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software," *in proceedings of the 12th Annual Network and Distributed System Security Symposium* (NDSS '05).

[10] Z. Li, D. Zou, S. Xu, X. Ou, H. Jin, S. Wang, Z. Deng, and Y. Zhong, "Vuldeepecker: A deep learning-based system for vulnerability detection," *arXiv preprint arXiv:1801.01681,* 2018.

[11] D. Zou, S. Wang, S. Xu, Z. Li, and H. Jin, "uvuldeepecker: A deep learning-based system for multiclass vulnerability detection," *IEEE Transactions on Dependable and Secure Computing,* 2019.

[12] Z. Li, D. Zou, S. Xu, H. Jin, Y. Zhu, and Z. Chen, "Sysevr: A framework for using deep learning to detect software vulnerabilities," *IEEE Transactions on Dependable and Secure Computing,* 2021.

[13] X. Duan, J. Wu, S. Ji, Z. Rui, T. Luo, M. Yang, and Y. Wu, "Vulsniper: Focus your attention to shoot fine-grained vulnerabilities," in *Proceedings of the TwentyEighth International Joint Conference on Artificial Intelligence,IJCAI-19,* pp. 4665-4671, 2019.

[14] C. Zhu, G. Du, T. Wu, N. Cui and L. Chen, "BERT-Based Vulnerability Type Identification with Effective Program Representation," *International Conference on Wireless Algorithms, Systems, and Applications, WASA 2022. Lecture Notes in Computer Science,* vol 13471. Springer, Cham.

[15] G. Lin, J. Zhang, W. Luo, L. Pan, and Y. Xiang, "Poster: Vulnerability discovery with function representation learning from unlabeled projects," *In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS),* pages 2539-2541. ACM, 2017.

[16] G. Lin, J. Zhang, W. Luo, L. Pan, O. Vel, P. Montague, and Y. Xiang, "Software vulnerability discovery via learning multi-domain knowledge bases," *IEEE Transactions on Dependable and Secure Computing,* 2019.

[17] V. Nguyen, T. Le, O. Vel, P. Montague, J. Grundy, and D. Phung, "Dualcomponent deep domain adaptation: A new approach for cross project software vulnerability detection," *PAKDD* 2020: 699-711.

[18] S. Liu, G. Lin, L. Qu, J. Zhang, O. Vel, P. Montague, and Y. Xiang, "Cd-vuld: Cross-domain vulnerability discovery based on deep domain adaptation," *IEEE Transactions on Dependable and Secure Computing,* 2020. doi: 10.1109/TDSC.2020.2984505.

[19] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang and T. Darrell, "Decaf: A deep convolutional activation feature for generic visual recognition." *In International conference on machine learning,* 647-655.

[20] A. Behera, Z, Hewage, and Bera,"Context-aware Attentional Pooling (CAP) for Fine-grained Visual Classification," *Proceedings of the AAAI Conference on Artificial Intelligence,* 35(2), 929-937.

[21] J. Zhang, X. Wang and X. Liu, "A Novel Neural Source Code Representation Based on Abstract Syntax Tree," *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), Montreal, QC, Canada,* 2019.

[22] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. "Greedy layer-wise training of deep networks." *In Advances in neural information processing systems,* pages 153-160,2007.

[23] B. Sun, J. Feng, and K. Saenko, "Return of frustratingly easy domain adaptation. In *AAAI, volume 6*," page 8, 2016.

[24] R. Sanodiya, J. Mathew, "A framework for semi-supervised metric transfer learning on manifolds." *Knowledge-Based Systems,*2019.

[25] "swftools," http://www.swftools.org/.