CROSS PROJECT SOFTWARE VULNERABILITY DETECTION VIA DOMAIN ADAPTATION AND MAX-MARGIN PRINCIPLE

Anonymous authors
Paper under double-blind review

ABSTRACT

Software vulnerabilities (SVs) have become a common, serious and crucial concern due to the ubiquity of computer software. Many machine learning-based approaches have been proposed to solve the software vulnerability detection (SVD) problem. However, there still have been two significant issues presenting for SVD in terms of i) learning automatic representations to improve the predictive performance of SVD, and ii) tackling the scarcity of labeled vulnerabilities datasets that conventionally need laborious labeling effort by experts. In this paper, we propose a novel end-to-end approach to tackle these two crucial issues. We first exploit the automatic representation learning with deep domain adaptation for software vulnerability detection. We then propose a novel cross-domain kernel classifier leveraging the max-margin principle to significantly improve the transfer learning process of software vulnerabilities from labeled projects into unlabeled ones. The experimental results on real-world software datasets show the superiority of our proposed method over state-of-the-art baselines.

1 INTRODUCTION

In the field of software security, software vulnerabilities (SVs), defined as specific flaws or oversights in software programs allowing attackers to exploit the code base and potentially undertake dangerous activities (e.g., exposing or altering sensitive information, disrupting, degrading or destroying a system, or taking control of a program or computer system) (Dowd et al., 2006), are very common and popular due to the ubiquity of computer software. Given the vast variety of technologies and different software development methodologies used, a great deal of computer software may naturally contain software vulnerabilities, hence making the problem of software vulnerability detection become a critical concern in computer security and software engineering. Moreover, the severity of the threat imposed by software vulnerabilities (SV) has significantly increased over the years causing significant damages to companies and individuals. As a result, this has necessitated the development of automated advanced approaches and tools that can efficiently and effectively detect SVs with a minimal level of human intervention. To respond to this demand, many vulnerability detection systems and methods, ranging from open source to commercial tools, and from manual to automatic methods (Shin et al., 2011; Neuhaus et al., 2007; Yamaguchi et al., 2011; Grieco et al., 2016; Li et al., 2016; Kim et al., 2017; Li et al., 2018a; Duan et al., 2019; Cheng et al., 2019; Zhuang et al., 2020) have been proposed and implemented.

Most previous work in software vulnerability detection (SVD) (Neuhaus et al., 2007; Shin et al., 2011; Yamaguchi et al., 2011; Li et al., 2016; Grieco et al., 2016; Kim et al., 2017) has been developed based on handcrafted features which are manually chosen by knowledgeable domain experts with possibly outdated experience and underlying biases. In many situations, handcrafted features normally do not generalize well. For example, features that work well in a certain software project may not perform well in other projects (Zimmermann et al., 2009). To alleviate the dependency on handcrafted features, the use of automatic features in SVD has been studied recently (Li et al., 2018a; Lin et al., 2018; Dam et al., 2018; Li et al., 2018b; Duan et al., 2019; Cheng et al., 2019; Zhuang et al., 2020). These works have shown the advantages of employing automatic features over handcrafted features in the context of software vulnerability detection.

Another major challenging issue in SVD research is the scarcity of labeled software projects to train models. The process of labeling vulnerable source code is tedious, time-consuming, error-prone, and can be very challenging even for domain experts. This has resulted in few labeled projects compared with a vast volume of unlabeled ones. Some recent approaches (Nguyen et al., 2019; 2020; Liu et al., 2020) have been proposed to solve this challenging problem with the aim to transfer the learning of vulnerabilities from labeled source domains to unlabeled target domains. Particularly, the methods in Nguyen et al. (2019; 2020) learn domain-invariant features from the code data of the source and target domains by using the adversarial learning framework such as Generative Adversarial Network (GAN) (Goodfellow et al., 2014) while the method in Liu et al. (2020) consists of many subsequent stages: i) pre-training a deep feature model for learning representation of token sequences (i.e., source code data), ii) learning cross-domain representations using a transformation to project token sequence embeddings from (i) to a latent space, and iii) training a classifier from the representations of the source domain data obtained from (ii). However, none of these methods exploit the imbalanced nature of source code projects for which the vulnerable codes are significantly minor comparing to non-vulnerable ones.

On the other side, kernel methods are well-known for their ability to deal with imbalanced datasets (Schölkopf et al., 2001; Tax & Duin, 2004; Tsang et al., 2005; 2007; Le et al., 2010; Nguyen et al., 2014). In a nutshell, a *domain of majority*, which is often a simple geometric shape in a feature space (e.g., half-hyperplane (Schölkopf et al., 2001; Nguyen et al., 2014) or hypersphere (Tax & Duin, 2004; Tsang et al., 2005; 2007; Le et al., 2010)) is defined to characterize the majority class (i.e., non-vulnerable codes). The key idea is that a simple domain of majority in the feature space, when being mapped back to the input space, forms a set of contours which can distinguish majority data from minority data.

In this paper, by leveraging learning domain-invariant features and kernel methods with the max-margin principle, we propose ***D**omain **A**daptation with **M**ax-**M**argin **P**rinciple* (DAM2P) to efficiently transfer learning from imbalanced labeled source domains to imbalanced unlabeled target domains. Particularly, inspired by the max-margin principle proven efficiently and effectively for learning from imbalanced data, when learning domain-invariant features, we propose to learn a max-margin hyperplane on the feature space to separate vulnerable and non-vulnerable data. More specifically, we combine labeled source data and unlabeled target data, and then learn a hyperplane to separate *source non-vulnerable from vulnerable data* and *target data from the origin* such that the margin is maximized. In addition, the margin is defined as the minimization of the source and target margins in which the *source margin* is regarded as the minimal distance from vulnerable data points to hyperplane (Nguyen et al., 2014), while the *target margin* is regarded as the distance from the origin to the hyperplane (Schölkopf et al., 2001). Our contributions can be summarized as follows:

- We leverage learning domain-invariant features and kernel methods with the max-margin principle to propose a novel approach named DAM2P which can efficiently bridge the gap between source and target domains on a joint space while being able to tackle efficiently and effectively the imbalanced nature of source and target domains.
- We conduct extensive experiments on real-world software datasets consisting of FFmpeg, LibTIFF, LibPNG, VLC, and Pidgin software projects. It is worth noting that to demonstrate and compare the capability of our proposed method and baselines in the transfer learning for SVD, the datasets (FFmpeg, VLC, and Pidgin) from the *multimedia application domain* are used as the source domains whilst the datasets (LibPNG and LibTIFF) from the *image application domain* were used as the target domains. The experimental results show that our method significantly outperforms the baselines by a wide margin, especially for the F1-measure.

2 Related Work

Automatic features in software vulnerability detection (SVD) has been widely studied (Li et al., 2018a; Lin et al., 2018; Dam et al., 2018; Li et al., 2018b; Duan et al., 2019; Cheng et al., 2019; Zhuang et al., 2020) due to its advantages of employing automatic features over handcrafted features in the context of SVD. Particularly, Dam et al. (2018) employed a deep neural network to transform sequences of code tokens to vectorial features further fed to a separate classifier while Li et al. (2018a) combined the learning of the vector representation and the training of the classifier in a

deep network. Advanced deep net architectures have been investigated for SVD problem. Notably, Russell et al. (2018) combined both recurrent neural networks (RNNs) and convolutional neural network (CNNs) for feature extraction from the embedded source code representations while Zhuang et al. (2020) proposed a new model for smart contract vulnerability detection using graph neural networks (GNNs).

Deep domain adaptation-based methods have been recently studied for SVD. Notably, Nguyen et al. (2019) proposed a novel architecture and employed adversarial learning framework (e.g., GAN) to learn domain-invariant features that can be transferred from labeled source to unlabeled target code project. Nguyen et al. (2020) enhanced Nguyen et al. (2019) by proposing an elegant workaround to combat the mode collapsing problem possibly faced in that work due to the use of GAN. Finally, Liu et al. (2020) proposed a multi-stage approach including three stages undertaken sequentially: i) pre-training a deep model for learning representation of token sequences (i.e., source code data), ii) learning cross-domain representations using a transformation to project token sequence embeddings from (i) to a latent space, iii) training a classifier from the representations of the source domain data obtained from (ii).

3 Domain Adaptation with Max-Margin Principle

3.1 Problem Formulation

Given a labeled source domain dataset $S = \left\{ \left(x_1^S, y_1\right), \ldots, \left(x_{N_S}^S, y_{N_S}\right) \right\}$ where $y_i \in \{0,1\}$ (i.e., 1: vulnerable code and 0: non-vulnerable code), let $x_i^S = \left[x_{i1}^S, \ldots, x_{iL}^S\right]$ be a code function represented as a sequence of L embedding vectors. We note that each embedding vector corresponds to a statement in the code function. Similarly, the unlabeled target domain dataset $T = \left\{x_1^T, \ldots, x_{N_T}^T\right\}$ consists the code functions $x_i^T = \left[x_{i1}^T, \ldots, x_{iL}^T\right]$ as sequences of L embedding vectors. For the data processing and embedding, please refer to the appendix section.

In standard DA approaches, domain-invariant features are learned on a joint space so that a classifier mainly trained based on source label data can be transferred to predict well unlabeled target data. The classifiers of interest are usually deep nets conducted on top of domain-invariant features. In this work, by leveraging with the kernel theory and the max-margin principle, we consider a kernel machine on top of domain-invariant features, which is a hyperplane on a feature space via a feature map ϕ. Inspired by the max-margin principle proven efficient and effective for learning from imbalanced data, when learning domain-invariant features, we propose to learn a max-margin hyperplane on the feature space to separate vulnerable and non-vulnerable codes. More specifically, we combine labeled source data and unlabeled target data, and then learn a hyperplane to separate *source non-vulnerable from vulnerable data* and *target data from the origin* such that the margin is maximized. Furthermore, the margin is defined as the minimization of the source and target margins in which the *source margin* is defined as the minimal distance from vulnerable data points to hyperplane (Nguyen et al., 2014) while the *target margin* is defined as the distance from the origin to the hyperplane (Schölkopf et al., 2001).

3.2 Our Proposed Approach DAM2P

3.2.1 Domain Adaptation for Learning Domain-Invariant Features

In what follows, we present the architecture of the generator G and how to use an adversarial learning framework such as GAN (Goodfellow et al., 2014) to learn domain-invariant features on a join space specified by G. To learn the automatic features for the sequential source code data, inspired from Li et al. (2018a); Nguyen et al. (2019; 2020), we apply a bidirectional recurrent neural network (bidirectional RNN) to both source and target domains. Given a code x in source or target domain, we denote the output of the bidirectional RNN by $\mathscr{B}(x)$. We then use some fully connected layers to connect the output layer of the bidirectional RNN with the joint feature layer wherein we bridge the gap between the source and target domains. The generator is consequently the composition of the bidirectional RNN and the subsequent fully connected layers: $G(x) = f(\mathscr{B}(x))$ where $f(\cdot)$ represents the map formed by the fully connected layers.

Subsequently, to bridge the gap between the source and target domains in the latent space, inspired by GAN (Goodfellow et al., 2014), we use a domain discriminator D to discriminate the source and target data and train the generator G to fool the discriminator D by making source and target data space indistinguishable in the latent. The relevant objective function is hence as follows:

$$\mathscr{H}(G,D) := \frac{1}{N_S}\sum_{i=1}^{N_S}\log D\left(G\left(x_i^S\right)\right) + \frac{1}{N_T}\sum_{i=1}^{N_T}\log\left[1-D\left(G\left(x_i^T\right)\right)\right] \quad (1)$$

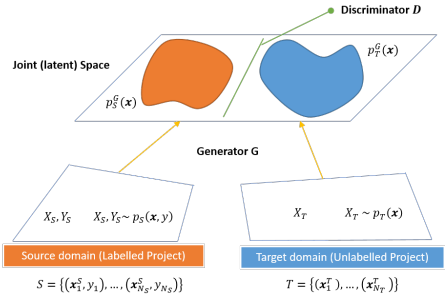

Figure 1: The architecture of using adversarial learning framework such as GAN to bridge the gap between the source and target domains in the latent space.

where we seek the optimal generator G^* and the domain discriminator D^* by solving:

$$G^* = \underset{G}{\operatorname{argmin}}\ \mathscr{H}(G,D) \text{ and } D^* = \underset{D}{\operatorname{argmax}}\ \mathscr{H}(G,D)$$

The architecture of using an adversarial learning framework such as GAN to bridge the gap between the source and target domains is depicted in Figure 1.

3.2.2 Cross-domain kernel classifier

To build up an efficient domain adaptation approach for source code data which can tackle well the imbalanced nature of source code projects, we leverage learning domain-invariant features with the max-margin principle in the context of kernel machines to propose a novel cross-domain kernel classifier named DAM2P. We construct a hyperplane on the feature space: $\mathbf{w}^T\phi(G(x)) - \rho = 0$ with the feature map ϕ and learn this hyperplane using the max-margin principle. More specifically, we combine labeled source and unlabeled target data, and then learn a hyperplane to separate *source non-vulnerable from vulnerable data* and *target data from the origin* in such a way that the margin is maximized. Moreover, the margin is defined as the minimization of the *source* and *target margins* in which the *source margin* is the minimal distance from vulnerable data points to hyperplane (Nguyen et al., 2014), while the *target margin* is the distance from the origin to the hyperplane (Schölkopf et al., 2001). The overall architecture of our proposed cross-domain kernel classifier in the feature space is depicted in Figure 2.

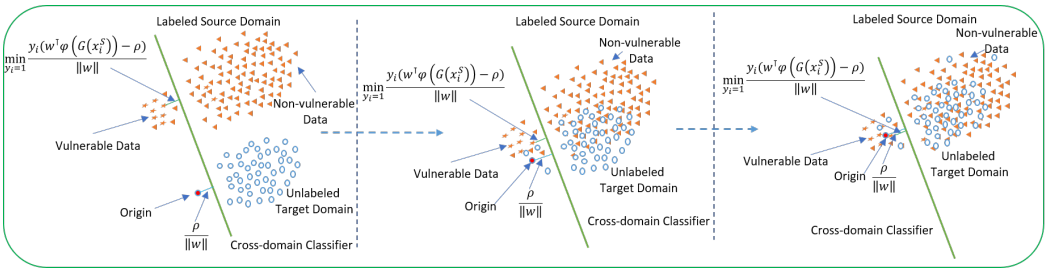

Figure 2: The architecture of our proposed cross-domain kernel classifier in the feature space. By using our proposed DAM2P method, we can gradually bridge the gap between the source and target domains in the latent space, while in the feature space our proposed cross-domain kernel classifier helps to distinguish the vulnerable and non-vulnerable data. At the end, when the source and target domains are intermingled, we can transfer our trained cross-domain kernel classifier to classify the data of the target domain.

Given the source domain dataset $S = \{(x_1^S, y_1), \ldots, (x_{N_S}^S, y_{N_S})\}$ where $y_i = 1, i = 1, ..., m$ and $y_i = 0, i = m+1, ..., N_S$ and the target domain dataset $T = \{x_1^T, \ldots, x_{N_T}^T\}$, we formulate the following optimization problem:

$$\max_{\mathbf{w},\rho}\left(\min\left\{\underbrace{\min_{y_i=1}\left\{\frac{y_i(\mathbf{w}^\top\phi(G(x_i^S))-\rho)}{\|\mathbf{w}\|}\right\}}_{\text{source margin}},\underbrace{\frac{\rho}{\|\mathbf{w}\|}}_{\text{target margin}}\right\}\right) \tag{2}$$

subject to

$$y_i(\mathbf{w}^\top\phi(G(x_i^S))-\rho)\geq 0, i=1,...,N_S$$
$$\mathbf{w}^\top\phi(G(x_i^T))\geq\rho, i=1,...,N_T.$$

In optimization problem (2), $\mathbf{w}$ and ρ are the normal vector and the bias of the hyperplane and ϕ is a transformation from the joint latent space to the feature space, while G is the generator used to map the data of source and target domains from the input space into the joint latent space. It occurs that the margin is invariant if we scale $\mathbf{w},\rho$ by a factor $k>0$. Hence without loosing of generality, we can assume that $\min\left\{\min_{y_i=1}\left\{y_i\mathbf{w}^\top\phi(G(x_i^S)-\rho\right\},\rho\right\}=1$[1]. The optimization problem (2) can be rewritten as follows:

$$\min_{w,\rho}\frac{1}{2}\|\mathbf{w}\|^2 \tag{3}$$

subject to

$$y_i(\mathbf{w}^\top\phi(G(x_i^S))-\rho)\geq 0, i=1,...,m$$
$$y_i(\mathbf{w}^\top\phi(G(x_i^S))-\rho)\geq 1, i=m+1,...,N_S$$
$$\mathbf{w}^\top\phi(G(x_i^T))\geq\rho, i=1,...,N_T$$

We refer the above model as hard version of our proposed cross-domain kernel classifier. To derive the soft version, we extend the optimization problem in Eq. (3) by using the slack variables as follows:

$$\min_{\mathbf{w},\rho}\left(\frac{1}{2}\|\mathbf{w}\|^2+\frac{1}{N_S+N_T}\left(\sum_{i=1}^{N_S}\xi_i^S+\lambda\sum_{i=1}^{N_T}\xi_i^T\right)\right) \tag{4}$$

subject to

$$y_i(\mathbf{w}^\top\phi(G(x_i^S))-\rho)\geq-\xi_i^S, i=1,...,m$$
$$y_i(\mathbf{w}^\top\phi(G(x_i^S))-\rho)\geq 1-\xi_i^S, i=m+1,...,N_S$$
$$\mathbf{w}^\top\phi(G(x_i^T))\geq\rho-\xi_i^T, i=1,...,N_T$$
$$\xi_i^S\geq 0, i=1,...,N_S; \xi_i^T\geq 0, i=1,...,N_T.$$

where $\lambda>0$ is the trade-off hyper-parameter representing the weight of the information from the target domain contributing to the cross-domain kernel classifier.

The primal form of the soft model optimization problem is hence of the following form:

$$\min_{\mathbf{w},\rho}\mathscr{L}(G,\mathbf{w},\rho) \tag{5}$$

where we have defined

$$\begin{aligned}\mathscr{L}(G,\mathbf{w},\rho):=&\frac{1}{2}\|\mathbf{w}\|^2+\frac{1}{N_S+N_T}\sum_{i=1}^{m}\max\left\{0,-y_i\left(\mathbf{w}^\top\phi(G(x_i^S)-\rho\right)\right\}\\&+\frac{1}{N_S+N_T}\sum_{i=m+1}^{N_S}\max\left\{0,-y_i(\mathbf{w}^\top\phi(G(x_i^S))-\rho)+1\right\}\\&+\frac{\lambda}{N_S+N_T}\sum_{i=1}^{N_T}\max\left\{0,-\mathbf{w}^\top\phi(G(x_i^T))+\rho\right\}\end{aligned}$$

[1]This assumption is feasible because if $(\mathbf{w}^*,\rho^*)$ is the optimal solution, $(k\mathbf{w}^*,k\rho^*)$ with $k>0$ is also another optimal solution. Therefore, we can choose k to satisfy the assumption.

We use a random feature map (Rahimi & Recht, 2008) for the transformation ϕ to map the representations (e.g., $G(x_i^S)$ and $G(x_i^T)$) from the latent space to a random feature space. The formulation of ϕ on a specific $G(x_i) \in \mathbb{R}^d$ is as follows:

$$\phi(G(x_i)) = \left[\frac{1}{\sqrt{K}} cos(\omega_k^\top G(x_i), \frac{1}{\sqrt{K}} sin(\omega_k^\top G(x_i)\right]_{k=1}^{K} \in \mathbb{R}^{2K}$$

where K consists of independent and identically distributed samples $\omega_1, ..., \omega_K \in \mathbb{R}^d$ which are the Fourier random elements.

We note that the use of a random feature map ϕ (Rahimi & Recht, 2008) in conjunction with the cost-sensitive kernel machine of our proposed cross-domain kernel classifier as mentioned in Eq. (5) and a bidirectional recurrent neural network for the generator G allows us to conveniently do back-propagation when training our proposed approach. Combining the optimization problems in Eqs. (1 and 5), we arrive at the final objective function:

$$\mathscr{I}(G, D, \mathbf{w}, \rho) := \mathscr{L}(G, \mathbf{w}, \rho) + \alpha \mathscr{H}(G, D) \tag{6}$$

where $\alpha > 0$ is the trade-off hyper-parameter. We seek the optimal generator G^*, domain discriminator D^*, the normal vector $\mathbf{w}^*$ and bias ρ^* by solving:

$$(G^*, \mathbf{w}^*, \rho^*) = \underset{G, \mathbf{w}, \rho}{\operatorname{argmin}} \mathscr{I}(G, D, \mathbf{w}, \rho) \text{ and } D^* = \underset{D}{\operatorname{argmax}} \mathscr{I}(G, D, \mathbf{w}, \rho)$$

4 EXPERIMENTS

Experimental Datasets We used the real-world datasets experimented in Nguyen et al. (2019; 2020) and collected by Lin et al. (2018). These contain the source code of vulnerable functions (vul-funcs) and non-vulnerable functions (non-vul-funcs) obtained from six real-world software project datasets, namely FFmpeg (#vul-funcs: 187 and #non-vul-funcs: 5427), LibTIFF (#vul-funcs: 81 and #non-vul-funcs: 695), LibPNG (#vul-funcs: 43 and #non-vul-funcs: 551), VLC (#vul-funcs: 25 and #non-vul-funcs: 5548), and Pidgin (#vul-funcs: 42 and #non-vul-funcs: 8268). These datasets cover multimedia and image application categories.

In the experiments, to demonstrate the capability of our proposed method in the transfer learning for software vulnerability detection (SVD) (i.e., transfering the learning of software vulnerabilities (SVs) from labelled projects to unlabelled projects belonging to different application domains), the datasets (FFmpeg, VLC, and Pidgin) from the multimedia application domains were used as the source domains, whilst the datasets (LibPNG and LibTIFF) from the image application domains were used as the target domains. It is worth noting that in the training process we hide the labels of datasets from the target domains. We only use these labels in the testing phase to evaluate the models' performance. Moreover, we used 80% of the target domain without labels in the training process, while the rest 20% was used for evaluating the domain adaptation performance. Note that these partitions were split randomly as used in the baselines. *Please refer to the appendix for details.*

Baselines The main baselines of our proposed DAM2P method are the state-of-the-art end-to-end deep DA approaches for SVD including DDAN (Ganin & Lempitsky, 2015), MMD (Long et al., 2015), D2GAN (Nguyen et al., 2017), DIRT-T (Shu et al., 2018), SCDAN (Nguyen et al., 2019), and Dual-GD-DDAN and Dual-GD-SDDAN (Nguyen et al., 2020) as well as the state-of-the-art automatic feature learning for SVD, VulDeePecker (Li et al., 2018a). To the method operated via separated stages proposed by Liu et al. (2020), at present, we cannot compare to it due to the lack of original data and completed reproducing source code from the authors.

VulDeePecker (Li et al., 2018a) is an automatic feature learning method for SVD. The model employed a bidirectional recurrent neural network to take sequential inputs and then concatenated hidden units as inputs to a feedforward neural network classifier while the DDAN, MMD, D2GAN and DIRT-T methods are the state-of-the-art deep domain adaptation models for computer vision proposed in Ganin & Lempitsky (2015), Long et al. (2015), Nguyen et al. (2017) and Shu et al. (2018) respectively. Inspired from Nguyen et al. (2019), we borrowed the principle of these methods and refactored them using the CDAN architecture introduced in Nguyen et al. (2019) for SVD.

The SCDAN method (Nguyen et al., 2019) can be considered as the first one demonstrating the feasibility of deep domain adaptation for SVD. Based on their proposed CDAN architecture, leveraging deep domain adaptation with automatic feature learning for SVD, the authors proposed the SCDAN method to efficiently exploit and utilize the information from unlabeled target data in order to improve the model performance. The Dual-GD-DDAN and Dual-GD-SDDAN methods proposed in Nguyen et al. (2020) aiming to deal with the mode collapsing problem faced in SCDAN and other approaches (i.e., using GAN as a principle in order to close the gap between source and target domains in the joint space) to further improve the transfer learning process for SVD.

For the data processing and embedding, and the model's configuration, please refer to the appendix.

Table 1: Performance results in terms of false negative rate (FNR), false positive rate (FPR), Recall, Precision and F1-measure of VulDeePecker (VULD), MMD, D2GAN, DIRT-T, DDAN, SCDAN, Dual-GD-DDAN, Dual-GD-SDDAN and DAM2P methods for predicting vulnerable and non-vulnerable code functions on the testing set of the target domain (Best performance in **bold**).

Source → Target	Methods	FNR	FPR	Recall	Precision	F1-measure
Pidgin → LibPNG	VULD	42.86%	1.08%	57.14%	80%	66.67%
	MMD	37.50%	**0%**	62.50%	**100%**	76.92%
	D2GAN	33.33%	1.06%	66.67%	80%	72.73%
	DIRT-T	33.33%	1.06%	66.67%	80%	72.73%
	DDAN	37.50%	**0%**	62.50%	**100%**	76.92%
	SCDAN	33.33%	**0%**	66.67%	100%	80%
	Dual-GD-DDAN	33.33%	**0%**	66.67%	**100%**	80%
	Dual-GD-SDDAN	22.22%	1.09%	77.78%	87.50%	82.35%
	DAM2P (ours)	**12.50%**	1.08%	**87.50%**	87.50%	**87.50%**
FFmpeg → LibTIFF	VULD	43.75%	6.72%	56.25%	50%	52.94%
	MMD	28.57%	12.79%	71.43%	47.62%	57.14%
	D2GAN	30.77%	6.97%	69.23%	64.29%	66.67%
	DIRT-T	25%	9.09%	75%	52.94%	62.07%
	DDAN	35.71%	6.98%	64.29%	60%	62.07%
	SCDAN	14.29%	5.38%	85.71%	57.14%	68.57%
	Dual-GD-DDAN	**12.5%**	8.2%	**87.5%**	56%	68.29%
	Dual-GD-SDDAN	35.29%	**3.01%**	64.71%	**73.33%**	68.75%
	DAM2P (ours)	14.29%	8.14%	85.71%	63.16%	**72.73%**
FFmpeg → LibPNG	VULD	25%	2.17%	75%	75%	75%
	MMD	12.5%	3.26%	87.5%	70%	77.78%
	D2GAN	14.29%	2.17%	85.71%	75%	80%
	DIRT-T	15.11%	2.2%	84.89%	80%	84.21%
	DDAN	**0%**	3.26%	**100%**	72.73%	84.21%
	SCDAN	12.5%	1.08%	87.5%	87.5%	87.5%
	Dual-GD-DDAN	**0%**	2.17%	**100%**	80%	88.89%
	Dual-GD-SDDAN	17.5%	**0%**	82.5%	100%	90.41%
	DAM2P (ours)	**0%**	1.07%	**100%**	**87.50%**	**93.33%**
VLC→ LibPNG	VULD	57.14%	**1.08%**	42.86%	75%	54.55%
	MMD	45%	4.35%	55%	60%	66.67%
	D2GAN	28.57%	4.3%	71.43%	55.56%	62.5%
	DIRT-T	50%	1.09%	50%	**80%**	61.54%
	DDAN	33.33%	2.20%	66.67%	75%	70.59%
	SCDAN	33.33%	1.06%	66.67%	80%	72.73%
	Dual-GD-DDAN	28.57%	2.15%	71.43%	71.43%	71.43%
	Dual-GD-SDDAN	**11.11%**	4.39%	**88.89%**	66.67%	76.19%
	DAM2P (ours)	33.33%	**0.00%**	66.67%	**100%**	**80%**
Pidgin → LibTIFF	VULD	35.29%	8.27%	64.71%	50%	56.41%
	MMD	30.18%	12.35%	69.82%	50%	58.27%
	D2GAN	40%	7.95%	60%	60%	60%
	DIRT-T	38.46%	8.05%	61.54%	53.33%	57.14%
	DDAN	27.27%	8.99%	72.73%	50%	59.26%
	SCDAN	30%	5.56%	70%	58.33%	63.64%
	Dual-GD-DDAN	29.41%	6.76%	70.59%	57.14%	63.16%
	Dual-GD-SDDAN	37.5%	**2.98%**	62.5%	**71.43%**	66.67%
	DAM2P (ours)	**7.69%**	9.20%	**92.31%**	60%	**72.73%**

4.1 EXPERIMENTAL RESULTS

4.1.1 CODE DOMAIN ADAPTATION FOR A FULLY NON-LABELED TARGET PROJECT

Quantitative Results We first investigated the performance of our proposed DAM2P method and compared it with the baselines. We note that the VulDeePecker method was only trained on the source data and then tested on the target data. The DDAN, MMD, D2GAN, DIRT-T, SCDAN, Dual-GD-DDAN, Dual-GD-SDDAN and DAM2P methods employed the target data without using any label information for domain adaptation.

The experimental results in Table 1 show that our proposed DAM2P method obtains a higher performance for almost measures in almost cases of the source and target domains. The DAM2P method always achieves the highest F1-measure for all pairs of the source and target domains. For example, in the case of the source domain (FFmpeg) and target domain (LibPNG), the DAM2P method obtains the F1-measure (93.33%) compared with the F1-measure (90.91%, 88.89%, 87.5%, 84.21%, 84.21%, 80%, 77.78% and 75%) obtained with Dual-GD-SDDAN, Dual-GD-DDAN, SCDAN, DDAN, DIRT-T, D2GAN, MMD and VulDeePecker, respectively.

Visualization We further demonstrate the efficiency of our proposed method in closing the gap of the source and target domains. We visualize the feature distributions of the source and target domains in the joint space using a t-SNE (Laurens & Geoffrey, 2008) projection with perplexity equal to 30. In particular, we project the source and target data in the joint space (i.e., $G(x)$) into a 2D space without undertaking domain adaptation (using the VulDeePecker method) and with undertaking domain adaptation (using our proposed DAM2P method).

In Figure 3, we observe these cases when performing domain adaptation from a software project (FFmpeg) to another (LibPNG). For the purpose of visualization, we select a random subset of the source project against the entire target project. As shown in Figure 3, without undertaking domain adaptation (VulDeePecker) the blue points (the source data) and the red points (the target data) are almost separate while with undertaking domain adaptation the blue and red points intermingled as expected. Furthermore, we observe that the mixing-up level of source and target data using our DAM2P method is significantly higher than using VulDeePecker. In this visualization, there is a strong correspondence between the success of domain adaptation regarding the classification accuracy of the target domain and the overlap between the domain distributions.

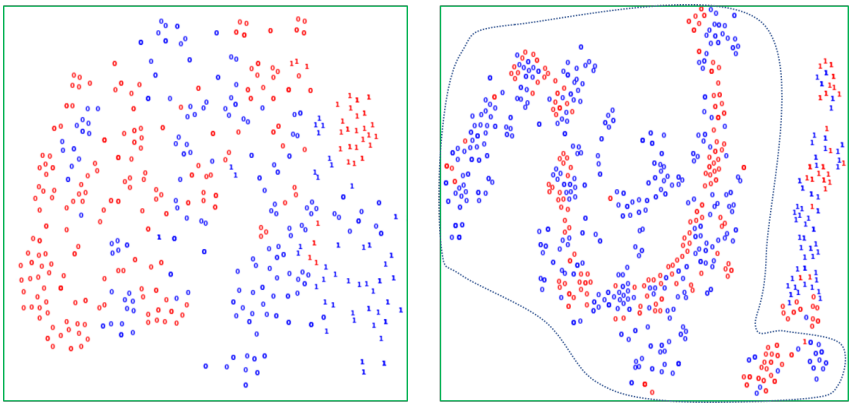

Figure 3: A 2D t-SNE projection for the case of the FFmpeg $\rightarrow$ LibPNG without undertaking domain adaptation (the left-hand figure, using the VulDeePecker method) and with undertaking domain adaptation (the right-hand figure, using our proposed DAM2P method). The blue and red points represent the source and target domains in the joint space respectively. In both cases of the source and target domains, data points labeled 0 stand for non-vulnerable samples and data points labeled 1 stand for vulnerable samples.

Ablation Study In this section, we aim to further demonstrate the efficiency of our DAM2P method in transferring the learning of software vulnerabilities from imbalanced labeled source domains to other imbalanced unlabeled target domains as well as the superiority of our novel cross-

domain kernel classifier in the our DAM2P method for learning and separating vulnerable and non-vulnerable data.

For this ablation study, we conduct experiments on two pairs FFmpeg → LibTIFF and FFmpeg → LibPNG. We want to demonstrate that briding the discrepancy gap in the latent space and the max-margin cross-domain kernel classifier are complementary to boost the domain adaptation performance with imbalanced nature. We consider five cases in which we start from the *blank case (i, VulDeePecker)* without briding gap and cross-domain kernel classifier. We then only add the GAN term to bridge the discrepancy gap in the *second case (ii, DDAN)*. In the *third case (iii, Kernel-Source)*, we only apply the max-margin principle for the source domain, while applying the max-margin principle for the source and target domains in the *fourth case (iv, Kernel-Source-Target)*. Finally, in the *last case (v, DAM2P)*, we simultaneously apply the bridging term and the max-margin terms for source and target domains. The results in Table 2 shows that the max-margin terms and bridging term help to boost the domain adaptation performance. Moreover, applying the max-margin term to both source and target domains improves the performance comparing to applying to only source domain. Last but not least, bridging the discrepancy gap term in cooperation with the max-margin term to significantly improve the domain adaptation performance.

Table 2: Performance results in terms of false negative rate (FNR), false positive rate (FPR), Recall, Precision and F1-measure of five cases including (i, VulDeePecker), (ii, DDAN), (iii, Kernel-Source), (iv, Kernel-Source-Target), and (v, DAM2P) for predicting vulnerable and non-vulnerable code functions on the testing set of the target domain (Best performance in **bold**).

Source → Target	Methods	FNR	FPR	Recall	Precision	F1-measure
FFmpeg → LibTIFF	VulDeePecker	43.75%	6.72%	56.25%	50%	52.94%
	DDAN	35.71%	6.98%	64.29%	60%	62.07%
	Kernel-Source	30%	**5.56%**	70%	58.33%	63.63%
	Kernel-Source-Target	25%	5.68%	75%	**64.29%**	69.23%
	DAM2P (ours)	14.29%	8.14%	**85.71%**	63.16%	**72.73%**
FFmpeg → LibPNG	VulDeePecker	25%	2.17%	75%	75%	75%
	DDAN	**0%**	3.26%	**100%**	72.73%	84.21%
	Kernel-Source	**0%**	4.39%	**100%**	69.23%	81.81%
	Kernel-Source-Target	**0%**	3.26%	**100%**	72.72%	84.21%
	DAM2P (ours)	**0%**	**1.07%**	**100%**	**87.50%**	**93.33%**

Please refer to the appendix section for the ablation study about the hyper-parameter sensitivity.

5 CONCLUSION

In this paper, in addition to exploiting deep domain adaptation with automatic representation learning for SVD, we have successfully proposed a novel cross-domain kernel classifier leveraging the max-margin principle to significantly improve the capability of the transfer learning of software vulnerabilities from labeled projects into unlabeled ones in order to deal with two crucial issues in SVD including i) learning automatic representations to improve the predictive performance of SVD, and ii) coping with the scarcity of labeled vulnerabilities in projects that require the laborious labeling of code by experts. Our proposed cross-domain kernel classifier can not only effectively deal with the imbalanced datasets problem in SVD but also leverage the information of the unlabeled projects to further improve the classifier's performance. The experimental results show the superiority of our proposed method compared with other state-of-the-art baselines in terms of the representation learning and transfer learning processes.

REFERENCES

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pp. 265–283, 2016.

Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16:321–357, 2002.

Xiao Cheng, Haoyu Wang, Jiayi Hua, Miao Zhang, Guoai Xu, Li Yi, and Yulei Sui. Static detection of control-flow-related vulnerabilities using graph embedding. In *2019 24th International Conference on Engineering of Complex Computer Systems (ICECCS)*, 2019.

Yin Cui, Menglin Jia, Tsung-Yi Lin, Yang Song, and Serge J. Belongie. Class-balanced loss based on effective number of samples. *CoRR*, abs/1901.05555, 2019.

H. K. Dam, T. Tran, T. Pham, N. S. Wee, J. Grundy, and A. Ghose. Automatic feature learning for predicting vulnerable software components. *IEEE Transactions on Software Engineering*, 2018.

M. Dowd, J. McDonald, and J. Schuh. *The Art of Software Security Assessment: Identifying and Preventing Software Vulnerabilities*. Addison-Wesley Professional, 2006. ISBN 0321444426.

Xu Duan, Jingzheng Wu, Shouling Ji, Zhiqing Rui, Tianyue Luo, Mutian Yang, and Yanjun Wu. Vulsniper: Focus your attention to shoot fine-grained vulnerabilities. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pp. 4665–4671, 2019.

Y. Ganin and V. Lempitsky. Unsupervised domain adaptation by backpropagation. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pp. 1180–1189, 2015.

I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pp. 2672–2680, 2014.

G. Grieco, G. L. Grinblat, L. Uzal, S. Rawat, J. Feist, and L. Mounier. Toward large-scale vulnerability discovery using machine learning. In *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy*, CODASPY '16, pp. 85–96, 2016. ISBN 978-1-4503-3935-3.

Sepp Hochreiter and Jűrgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8): 1735–1780, 1997.

S. Kim, S. Woo, H. Lee, and H. Oh. VUDDY: A scalable approach for vulnerable code clone discovery. In *IEEE Symposium on Security and Privacy*, pp. 595–614. IEEE Computer Society, 2017.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

V. M. Laurens and H. Geoffrey. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

T. Le, D. Tran, W. Ma, and D. Sharma. An optimal sphere and two large margins approach for novelty detection. In *Neural Networks (IJCNN), The 2010 International Joint Conference on*, pp. 1–6, July 2010.

Z. Li, D. Zou, S. Xu, H. Jin, H. Qi, and J. Hu. Vulpecker: An automated vulnerability detection system based on code similarity analysis. In *Proceedings of the 32Nd Annual Conference on Computer Security Applications*, ACSAC '16, pp. 201–213, 2016. ISBN 978-1-4503-4771-6.

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. *CoRR*, abs/1801.01681, 2018a.

Zhen Li, Deqing Zou, Shouhuai Xu, Hai Jin, Yawei Zhu, Zhaoxuan Chen, Sujuan Wang, and Jialai Wang. Sysevr: A framework for using deep learning to detect software vulnerabilities. *CoRR*, abs/1807.06756, 2018b.

G. Lin, J. Zhang, W. Luo, L. Pan, Y. Xiang, D. V. Olivier, and M. Paul. Cross-project transfer representation learning for vulnerable function discovery. In *IEEE Transactions on Industrial Informatics*, volume 14, 2018.

Tsung-Yi Lin, Priya Goyal, Ross B. Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. *CoRR*, abs/1708.02002, 2017.

Shigang Liu, Guanjun Lin, Lizhen Qu, Jun Zhang, Olivier De Vel, Paul Montague, and Yang 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.

M. Long, Y. Cao, J. Wang, and M. Jordan. Learning transferable features with deep adaptation networks. In F. Bach and D. Blei (eds.), *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 97–105, Lille, France, 2015.

S. Neuhaus, T. Zimmermann, C. Holler, and A. Zeller. Predicting vulnerable software components. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, CCS '07, pp. 529–540, 2007. ISBN 978-1-59593-703-2.

T. D. Nguyen, T. Le, H. Vu, and D. Q. Phung. Dual discriminator generative adversarial nets. *CoRR*, abs/1709.03831, 2017.

V. Nguyen, T. Le, T. Pham, M. Dinh, and T. H. Le. Kernel-based semi-supervised learning for novelty detection. In *2014 International Joint Conference on Neural Networks (IJCNN)*, pp. 4129–4136, July 2014.

V. Nguyen, T. Le, T. Le, K. Nguyen, O. DeVel, P. Montague, L. Qu, and D. Phung. Deep domain adaptation for vulnerable code function identification. In *The International Joint Conference on Neural Networks (IJCNN)*, 2019.

Van Nguyen, Trung Le, Olivier De Vel, Paul Montague, John Grundy, and Dinh Phung. Dual-component deep domain adaptation: A new approach for cross project software vulnerability detection. 2020.

Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In J. Platt, D. Koller, Y. Singer, and S. Roweis (eds.), *Advances in Neural Information Processing Systems*, volume 20. Curran Associates, Inc., 2008.

Rebecca L. Russell, Louis Y. Kim, Lei H. Hamilton, Tomo Lazovich, Jacob A. Harer, Onur Ozdemir, Paul M. Ellingwood, and Marc W. McConley. Automated vulnerability detection in source code using deep representation learning. *CoRR*, abs/1807.04320, 2018.

B. Schölkopf, J. C. Platt, J. C. Shawe-Taylor, A. J. Smola, and R. C. Williamson. Estimating the support of a high-dimensional distribution. *Neural Comput.*, 13(7):1443–1471, July 2001. ISSN 0899-7667.

Y. Shin, A. Meneely, L. Williams, and J A Osborne. Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities. *IEEE Transactions on Software Engineering*, 37(6):772–787, 2011.

R. Shu, H. Bui, H. Narui, and S. Ermon. A DIRT-t approach to unsupervised domain adaptation. In *International Conference on Learning Representations*, 2018.

D. M. J. Tax and R. P. W. Duin. Support vector data description. *Journal of Machine Learning Research*, 54(1):45–66, 2004.

I. W. Tsang, J. T. Kwok, P. Cheung, and N. Cristianini. Core vector machines: Fast svm training on very large data sets. *Journal of Machine Learning Research*, 6:363–392, 2005.

I. W. Tsang, A. Kocsor, and J. T. Kwok. Simpler core vector machines with enclosing balls. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, pp. 911–918, 2007.

F. Yamaguchi, F. Lindner, and K. Rieck. Vulnerability extrapolation: assisted discovery of vulnerabilities using machine learning. In *Proceedings of the 5th USENIX conference on Offensive technologies*, pp. 13–23, 2011.

Yuan Zhuang, Zhenguang Liu, Peng Qian, Qi Liu, Xiang Wang, and Qinming He. Smart contract vulnerability detection using graph neural network. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pp. 3283–3290, 2020.

T. Zimmermann, N. Nagappan, H. Gall, E. Giger, and B. Murphy. Cross-project defect prediction: A large scale experiment on data vs. domain vs. process. In *Proceedings of the the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, ESEC/FSE '09, pp. 91–100, 2009. ISBN 978-1-60558-001-2.

A Appendix

Data Processing and Embedding

We preprocess datasets before inputting them into the deep neural networks (i.e., baselines and our proposed method). Inspired from Nguyen et al. (2019; 2020), we first standardize the source code by removing comments, blank lines and non-ASCII characters. Secondly, we map user-defined variables to symbolic names (e.g., "*var1*", "*var2*") and user-defined functions to symbolic names (e.g., "*func1*", "*func2*"). We replace integers, real and hexadecimal numbers with a generic *<number>* token and strings with a generic *<str>* token. Thirdly, we embed statements in source code into vectors. In particular, each statement x_{ij} consists of two parts: the opcode and the statement information. We embed both opcode and statement information to vectors, then concatenate the vector representations of opcode and statement information to obtain the final vector representation $\mathbf{i}_{ij}$ of statement x_{ij}. For example, in the following statement *"if(func2(func3(number,number),&var2) !=var10)*", the opcode is *if* and the statement information is *(func2(func3(number,number), &var2)!=var10).* To embed the opcode, we multiply the one-hot vector of the opcode by the opcode embedding matrix. To embed the statement information, we tokenize it to a sequence of tokens (e.g., *(,func2,(,func3,(,number,number,),&,var2,),!=,var10,)*), construct the frequency vector of the statement information, and multiply this frequency vector by the statement information embedding matrix. This embedding process is depicted in Figure 4. In addition, the opcode embedding W^{op} and statement information embedding W^{si} matrices are learnable variables in the model.

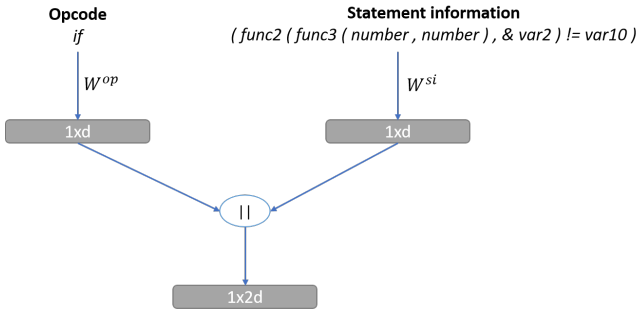

Figure 4: An example of a source code statement embedding process.

Figure 5 shows an example of the procedure for data processing and embedding. The embedding vectors (e.g., $[i_{i1}, \ldots, i_{iL}]$), which are obtained from data processing and embedding process, of each function (e.g., x_i can be from the source domain or the target domain) represent for the function to be the input to deep learning models (e.g., the baselines and our proposed method).

Note that as the baseline methods, to our proposed method, for handling the sequential properties of the data and to learn the automatic features of the source code functions, we also use a bidirectional recurrent neural network (bidirectional RNN) for both source and target domains.

Figure 5: An example of the procedure for data processing and embedding. We use a source code function in the C language programming from the FFmpeg project. After the data preprocessing step, we obtain a preprocessed function, and then using the embedding process we obtain the embedded vectors corresponding to statements of the function.

MODEL CONFIGURATION

For the baselines including VulDeePecker (Li et al., 2018a), and DDAN (Ganin & Lempitsky, 2015), MMD (Long et al., 2015), D2GAN (Nguyen et al., 2017), DIRT-T (Shu et al., 2018), SCDAN (Nguyen et al., 2019) using the architecture CDAN proposed in Nguyen et al. (2019), and Dual-GD-DDAN and Dual-GD-SDDAN (Nguyen et al., 2020), and our proposed DAM2P method, we use one bidirectional recurrent neural network with LSTM (Hochreiter & Schmidhuber, 1997) cells where the size of hidden states is in $\{128, 256, 512\}$ for the generator G while to the source classifier C used in the baselines and the domain discriminator D, we use deep feed-forward neural networks consisting of two hidden layers where the size of each hidden layer is in $\{200, 300\}$. We embed the opcode and statement information in the $\{150, 150\}$ dimensional embedding spaces respectively. The trade-off parameters λ and α of our proposed method are in $\{10^{-3}, 10^{-2}, 10^{-1}\}$ while the hidden size h is in $\{128, 256, 512\}$. The dimension of random feature space $2K$ is set equal to 512. The length L of each function is padded or cut to 100 code statements (i.e., We base on the quantile values of the functions' length of each dataset to decide the length of each function padded or cut to 100 code statements. Almost all important information relevant to the vulnerability lies in the 100 first code statements.)

We employed the Adam optimizer (Kingma & Ba, 2014) with an initial learning rate of 10^{-3} while the mini-batch size is set to 100 to our proposed method and baselines. We split the data of the source domain into two random partitions containing 80% for training and 20% for validation. We also split the data of the target domain into two random partitions. The first partition contains 80% for training the models of MMD, D2GAN, DIRT-T, DDAN, SCDAN, Dual-GD-DDAN, Dual-GD-SDDAN, and DAM2P without using any label information while the second partition contains 20% for testing the models. We additionally apply gradient clipping regularization to prevent the over-fitting problem in the training process of each model. We implement all mentioned methods in Python using Tensorflow (Abadi et al., 2016), an open-source software library for Machine Intelligence developed by the Google Brain Team.

ADDITIONAL EXPERIMENTS

Hyper-parameter Sensitivity In this section, we investigate the correlation between important hyper-parameters (including the λ, α, and h (the size of hidden states in the bidirectional neural network)) and the F1-measure of our proposed DAM2P method. As mentioned in the experiments section, the trade-off parameters λ and α are in $\{10^{-3}, 10^{-2}, 10^{-1}\}$ while the hidden size h is in

$\{128, 256, 512\}$. It is worth noting that we use the commonly used values for the trade-off hyper-parameters (λ and α) representing for the weights of different terms mentioned in Eq. (6) and the hidden size h. In order to study the impact of the hyper-parameters on the performance of the DAM2P method, we use a wider range of values for λ, α, and h. In this ablation study, the trade-off parameters λ and α are in $\{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 10^{0}, 10^{1}\}$ while the hidden size h is in $\{32, 64, 128, 256, 512, 1024\}$.

We investigate the impact of λ, α and h hyper-parameters on the performance of the DAM2P method on five pairs of the source and target domains including FFmpeg to LibPNG, FFmpeg to LibTIFF, Pidgin to LibPNG, Pidgin to LibTIFF, and VLC to LibPNG. As shown in Figures 6 and 7, we observe that the appropriate values to the hyper-parameters used in the DAM2P model in order to obtain the best model's performance should be in from 10^{-4} to 10^{-2}, from 10^{-3} to 10^{-1}, and from 64 to 256 for λ, α and h respectively. In particular, for the hidden size h, if we use too small values (e.g., ≤ 32) or too high values (e.g., ≥ 1024), the model might encounter the underfitting or overfitting problems respectively. The model's performance on λ (i.e., representing the weight of the information from the target domain contributing to the cross-domain kernel classifier during the training process) shows that we should not set the value of λ equal or higher than 1.0 (i.e., used for the weight of the information from the source domain), and the value of λ should be higher than 10^{-4} to make sure that we use enough information of the target domain in the training process to improve the cross-domain kernel classifier.

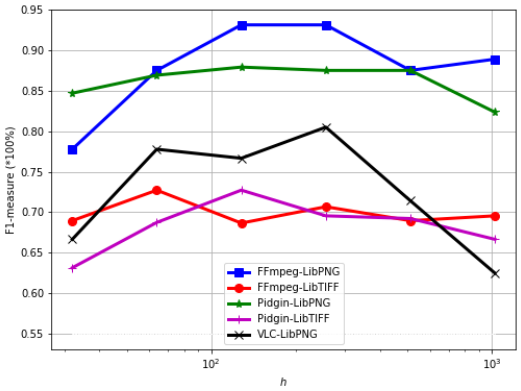

Figure 6: The correlation between h and F1-measure of our proposed DAM2P method.

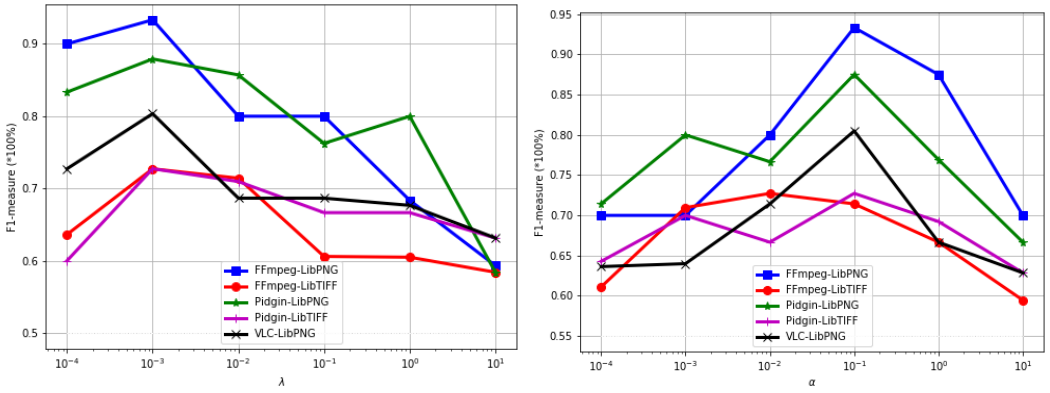

Figure 7: The correlation between (λ and α) and F1-measure of our proposed DAM2P method.

Additional ablation studies

Sampling and weighting are well-known as simple and heuristic methods to deal with imbalanced datasets. However, as mentioned by Lin et al. (2017); Cui et al. (2019), these methods have some limitations, for example, i) Sampling may either introduce large amounts of duplicated samples, which slows down the training and makes the model susceptible to over-fitting when oversampling, or discard valuable examples that are important for feature learning when under-sampling, and ii) To the highly imbalanced datasets, directly training the model or weighting (e.g., inverse class frequency or the inverse square root of class frequency) cannot yield satisfactory performance.

To experience these findings, we run an experiment on two pairs of the source and target domains (i.e., FFmpeg $\rightarrow$ LibTIFF and FFmpeg $\rightarrow$ LibPNG) for some baselines including DDAN, SCDAN, Dual-GD-DDAN, and Dual-GD-SDDAN methods using the over-sampling technique based on SMOTE (Chawla et al., 2002) (i.e., used to create balanced datasets). The experimental results

in Table 3 show that using the over-sampling technique cannot help improve these models' performance. In particular, the performance of these methods without using over-sampling is always higher than using over-sampling on the used datasets in F1-measure, the most important measure used in software vulnerability detection. Note that in Table 3, the term OS stands for over-sampling.

Table 3: Performance results in terms of false negative rate (FNR), false positive rate (FPR), Recall, Precision and F1-measure of DDAN, SCDAN, Dual-GD-DDAN and Dual-GD-SDDAN methods in two cases of using over-sampling (OS) and without using over-sampling (OS) for predicting vulnerable and non-vulnerable code functions on the testing set of the target domain.

Source → Target	Methods	FNR	FPR	Recall	Precision	F1-measure
FFmpeg → LibTIFF	DDAN with OS	50%	4.55%	50%	60%	54.55%
	DDAN without OS	35.71%	6.98%	64.29%	60%	62.07%
	SCDAN with OS	27.27%	7.87%	72.72%	55.33%	61.54%
	SCDAN without OS	14.29%	5.38%	85.71%	57.14%	68.57%
	Dual-GD-DDAN with OS	25%	6.72%	75%	57.14%	64.87%
	Dual-GD-DDAN without OS	12.5%	8.2%	87.5%	56%	68.29%
	Dual-GD-SDDAN with OS	16.67%	9.1%	83.33%	56%	67%
	Dual-GD-SDDAN without OS	35.29%	3.01%	64.71%	73.33%	68.75%
FFmpeg → LibPNG	DDAN with OS	14.29%	2.15%	85.71%	75%	80%
	DDAN without OS	0%	3.26%	100%	72.73%	84.21%
	SCDAN with OS	0%	4.4%	100%	69.23%	81.82%
	SCDAN without OS	12.5%	1.08%	87.5%	87.5%	87.5%
	Dual-GD-DDAN with OS	0%	2.15%	100%	77.78%	87.5%
	Dual-GD-DDAN without OS	0%	2.17%	100%	80%	88.89%
	Dual-GD-SDDAN with OS	0%	2.17%	100%	80%	88.89%
	Dual-GD-SDDAN without OS	17.5%	0%	82.5%	100%	90.41%