DUAL-COMPONENT DEEP DOMAIN ADAPTATION: A NEW APPROACH FOR CROSS PROJECT SOFTWARE VULNERABILITY DETECTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Owing to the ubiquity of computer software, software vulnerability detection (SVD) has become an important problem in the software industry and in the field of computer security. One of the most crucial issues in SVD is coping with the scarcity of labeled vulnerabilities in projects that require the laborious manual labeling of code by software security experts. One possible way to address is to employ deep domain adaptation which has recently witnessed enormous success in transferring learning from structural labeled to unlabeled data sources. The general idea is to map both source and target data into a joint feature space and close the discrepancy gap of those data in this joint feature space. Generative adversarial network (GAN) is a technique that attempts to bridge the discrepancy gap and also emerges as a building block to develop deep domain adaptation approaches with state-of-the-art performance. However, deep domain adaptation approaches using the GAN principle to close the discrepancy gap are subject to the mode collapsing problem that negatively impacts the predictive performance. Our aim in this paper is to propose Dual Generator-Discriminator Deep Code Domain Adaptation Network (Dual-GD-DDAN) for tackling the problem of transfer learning from labeled to unlabeled software projects in the context of SVD in order to resolve the mode collapsing problem faced in previous approaches. The experimental results on real-world software projects show that our proposed method outperforms state-of-the-art baselines by a wide margin.

1 INTRODUCTION

In the software industry, software vulnerabilities relate to specific flaws or oversights in software programs which allow attackers to expose or alter sensitive information, disrupt or destroy a system, or take control of a program or computer system (Dowd et al., 2006). The software vulnerability detection problem has become an important issue in the software industry and in the field of computer security. Computer software development employs of a vast variety of technologies and different software development methodologies, and much computer software contains vulnerabilities.

This has necessitated the development of automated advanced techniques and tools that can efficiently and effectively detect software vulnerabilities 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 have been proposed and implemented. Most of the previous works in software vulnerability detection (SVD) (Neuhaus et al., 2007; Shin et al., 2011; Yamaguchi et al., 2011; Almorsy et al., 2012; Li et al., 2016; Grieco et al., 2016; Kim et al., 2017) have been developed based on handcrafted features which are manually chosen by knowledgeable domain experts who may have 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., 2018; Lin et al., 2018; Dam et al., 2018). These works have shown the advantages of automatic features over handcrafted features in the context of software vulnerability detection.

However, most of these approaches lead to another crucial issue in SVD research, namely the scarcity of labeled projects. Labelled vulnerable code is needed to train these models, and the process of labeling vulnerable source code is very tedious, time-consuming, error-prone, and challenging even for domain experts. This has led to few labeled projects compared with the vast volume of unlabeled ones. A viable solution is to apply transfer learning or domain adaptation which aims to devise automated methods that make it possible to transfer a learned model from the source domain with labels to the target domains without labels. Studies in domain adaptation can be broadly categorized into two themes: shallow (Borgwardt et al., 2006; Gopalan et al., 2011) and deep domain adaptations (Ganin & Lempitsky, 2015; Tzeng et al., 2015; Long et al., 2015; Shu et al., 2018; French et al., 2018). These recent studies have shown the advantages of deep over shallow domain adaptation (i.e., higher predictive performance and capacity to tackle structural data). Deep domain adaptation encourages the learning of new representations for both source and target data in order to minimize the divergence between them (Ganin & Lempitsky, 2015; Tzeng et al., 2015; Long et al., 2015; Shu et al., 2018; French et al., 2018). The general idea is to map source and target data to a joint feature space via a generator, where the discrepancy between the source and target distributions is reduced. Notably, the work of (Ganin & Lempitsky, 2015; Tzeng et al., 2015; Shu et al., 2018) employed generative adversarial networks (GANs) (Goodfellow et al., 2014) to close the discrepancy gap between source and target data in the joint space. However, most of aforementioned works mainly focus on transfer learning in the computer vision domain. The work of (Nguyen et al., 2019) is the first work which applies deep domain adaptation to SVD with promising predictive performance on real-world source code projects. The underlying idea is to employ the GAN to close the gap between source and target domain in the joint space and enforce the clustering assumption (Chapelle & Zien, 2005) to utilize the information carried in the unlabeled target samples in a semi-supervised context.

GANs are known to be affected by the mode collapsing problem (Goodfellow, 2016; Santurkar et al., 2018). In particular, (Santurkar et al., 2018) recently studied the mode collapsing problem and further classified this into the missing mode problem i.e., the generated samples miss some modes in the true data, and the boundary distortion problem i.e., the generated samples can only partly recover some modes in the true data. It is certain that deep domain adaptation approaches that use the GAN principle will inherently encounter both the missing mode and boundary distortion problems. Last but not least, deep domain adaptation approaches using the GAN principle also face the data distortion problem. The representations of source and target examples in the joint feature space degenerate to very small regions that cannot preserve the manifold/clustering structure in the original space.

Our aim in this paper is to address not only deep domain adaptation mode collapsing problems but also boundary distortion problems when employing the GAN as a principle in order to close the discrepancy gap between source and target data in the joint feature space. Our two approaches are: i) apply manifold regularization for enabling the preservation of manifold/clustering structures in the joint feature space, hence avoiding the degeneration of source and target data in this space; and ii) invoke dual discriminators in an elegant way to reduce the negative impacts of the missing mode and boundary distortion problems in deep domain adaptation using the GAN principle as mentioned before. We name our mechanism when applied to SVD as *Dual Generator-Discriminator Deep Code Domain Adaptation Network* (Dual-GD-DDAN). We empirically demonstrate that our Dual-GD-DDAN can overcome the missing mode and boundary distortion problems which is likely to happen as in Deep Code Domain Adaptation (DDAN) (Nguyen et al., 2019) in which the GAN was solely applied to close the gap between the source and target domain in the joint space (see the discussion in Sections 2.4 and 3.3, and the visualization in Figure 3). In addition, we incorporate the relevant approaches – minimizing the conditional entropy and manifold regularization with spectral graph – proposed in (Nguyen et al., 2019) to enforce the clustering assumption (Chapelle & Zien, 2005) and arrive at a new model named *Dual Generator-Discriminator Semi-supervised Deep Code Domain Adaptation Network* (Dual-GD-SDDAN). We further demonstrate that our Dual-GD-SDDAN can overcome the mode collapsing problem better than SCDAN in (Nguyen et al., 2019), hence obtaining better predictive performance.

We conducted experiments using the data sets collected by (Lin et al., 2018), that consist of five real-world software projects: FFmpeg, LibTIFF, LibPNG, VLC and Pidgin to compare our proposed Dual-GD-DDAN and Dual-GD-SDDAN with the baselines. The baselines consider to include VULD (i.e., the model proposed in (Li et al., 2018) without domain adaptation), MMD, DIRT-

T, DDAN and SCDAN as mentioned (Nguyen et al., 2019) and D2GAN (Nguyen et al., 2017) (a variant of the GAN using dual-discriminator to reduce the mode collapse for which we apply this mechanism in the joint feature space). Our experimental results show that our proposed methods are able to overcome the negative impact of the missing mode and boundary distortion problems inherent in deep domain adaptation approaches when solely using the GAN principle as in DDAN and SCDAN (Nguyen et al., 2019). In addition, our method outperforms the rival baselines in terms of predictive performance by a wide margin.

2 Deep Code Domain Adaptation with GAN

2.1 Problem Statement

A source domain data set $S = \left\{ \left(\boldsymbol{x}_1^S, y_1 \right), \ldots, \left(\boldsymbol{x}_{N_S}^S, y_{N_S} \right) \right\}$ where $y_i \in \{-1, 1\}$ (i.e., 1: vulnerable code and -1: non-vulnerable code) and $\boldsymbol{x}_i^S = \left[\boldsymbol{x}_{i1}^S, \ldots, \boldsymbol{x}_{iL}^S \right]$ is a sequence of L embedding vectors, and the target domain data set $T = \left\{ \boldsymbol{x}_1^T, \ldots, \boldsymbol{x}_{N_T}^T \right\}$ where $\boldsymbol{x}_i^T = \left[\boldsymbol{x}_{i1}^T, \ldots, \boldsymbol{x}_{iL}^T \right]$ is also a sequence of L embedding vectors. We wish to bridge the gap between the source and target domains in the joint feature space. This allows us to transfer a classifier trained on the source domain to predict well on the target domain.

2.2 Data Processing and Embedding

We preprocess data sets before inputting into the deep neural networks. Firstly, we 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 also replace integers, real and hexadecimal numbers with a generic <*num*> token and strings with a generic <*str*> token. Thirdly, we embed statements in source code into vectors. In particular, each statement $\boldsymbol{x}$ 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 $\mathbf{i}$ of statement $\boldsymbol{x}$. For example, in the following statement (C programming language) *"if(func3(func4(num,num),&var2)!=var11)"*, the opcode is *if* and the statement information is *(func3(func4(num,num),&var2)!=var11)*. 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., *(,func3,(,func4,(,num,num,),&,var2,),!=,var11,))*, construct the frequency vector of the statement information, and multiply this frequency vector by the statement information embedding matrix. In addition, the opcode embedding and statement embedding matrices are learnable variables.

2.3 Deep Code Domain Adaptation with a Bidirectional RNN

To handle sequential data in the context of domain adaptation of software vulnerability detection, the work of (Nguyen et al., 2019) proposed an architecture referred to as the Code Domain Adaptation Network (CD*A*N). This network architecture recruits a Bidirectional RNN to process the sequential input from both source and target domains (i.e., $\boldsymbol{x}_i^S = \left[\boldsymbol{x}_{i1}^S, \ldots, \boldsymbol{x}_{iL}^S \right]$ and $\boldsymbol{x}_i^T = \left[\boldsymbol{x}_{i1}^T, \ldots, \boldsymbol{x}_{iL}^T \right]$). A fully connected layer is then employed to connect the output layer of the Bidirectional RNN with the joint feature layer while bridging the gap between the source and target domains. Furthermore, inspired by the Deep Domain Adaptation approach (Ganin & Lempitsky, 2015), the authors employ the source classifier $\mathcal{C}$ to classify the source samples, the domain discriminator D to distinguish the source and target samples and propose Deep Code Domain Adaptation (DDAN) whose objective function is as follows:

$$\mathcal{J}\left(G, D, C\right) = \frac{1}{N_S} \sum_{i=1}^{N_S} \ell\left(C\left(G\left(\boldsymbol{x}_i^S \right) \right), y_i \right) + \lambda \left(\frac{1}{N_S} \sum_{i=1}^{N_S} \log D\left(G\left(\boldsymbol{x}_i^S \right) \right) + \frac{1}{N_T} \sum_{i=1}^{N_T} \log \left[1 - D\left(G\left(\boldsymbol{x}_i^T \right) \right) \right] \right) \tag{1}$$

where seeking the optimal generator G^*, the domain discriminator D^*, and the source classifier C^* is found by solving:

$$\left(C^*, G^*\right) = \underset{C,G}{\operatorname{argmin}}\, \mathcal{J}\left(G, D, C\right) \text{ and } D^* = \underset{D}{\operatorname{argmax}}\, \mathcal{J}\left(G, D, C\right)$$

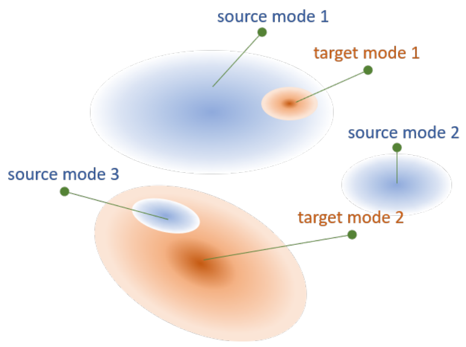


Figure 1: An illustration of the missing mode and boundary distortion problems of DDAN. In the joint space, the target distribution misses source mode 2, while the source distribution can only partly cover the target mode 2 in the target distribution and the target distribution can only partly cover the source mode 1 in the source distribution.

2.4 The shortcomings of DDAN

We observe that DDAN suffers from several shortcomings. First, the *data distortion* problem (i.e., the source and target data in the joint space might collapse into small regions) may occur since there is no mechanism in DDAN to circumvent this. Second, since DDAN is based on the GAN approach, DDAN might suffer from the mode collapsing problem (Goodfellow, 2016; Santurkar et al., 2018). In particular, (Santurkar et al., 2018) has recently studied the mode collapsing problem of GANs and discovered that they are also subject to i) the *missing mode* problem (i.e., in the joint space, either the target data misses some modes in the source data or vice versa) and ii) the *boundary distortion* problem (i.e., in the joint space either the target data partly covers the source data or vice versa), which makes the target distribution significantly diverge from the source distribution. As shown in Figure 1, both the missing mode and boundary distortion problems simultaneously happen since the target distribution misses source mode 2, while the source distribution can only partly cover the target mode 2 in the target distribution and the target distribution can only partly cover the source mode 1 in the source distribution.

3 Our Approach: Dual Generator-Discriminator Deep Code Domain Adaptation

3.1 Key Idea of Our Approach

We employ two discriminators (namely, D_S and D_T) to classify the source and target examples and vice versa and two separate generators (namely, G_S and G_T) to map the source and target examples to the joint space respectively. In particular, D_S produces high values on the source examples in the joint space (i.e., $G_S\left(\boldsymbol{x}^S\right)$) and low values on the target examples in the joint space (i.e., $G_T\left(\boldsymbol{x}^T\right)$), while D_T produces high values on the target examples in the joint space (i.e., $G_T\left(\boldsymbol{x}^T\right)$) and low values on the source examples (i.e., $G_S\left(\boldsymbol{x}^S\right)$). The generator G_S is trained to push $G_S\left(\boldsymbol{x}^S\right)$ to the high value region of D_T and the generator G_T is trained to push $G_T\left(\boldsymbol{x}^T\right)$ to the high value region of D_S. Eventually, both $D_S\left(G_S\left(\boldsymbol{x}^S\right)\right)$ and $D_S\left(G_T\left(\boldsymbol{x}^T\right)\right)$ are possibly high and both $D_T\left(G_S\left(\boldsymbol{x}^S\right)\right)$ and $D_T\left(G_T\left(\boldsymbol{x}^T\right)\right)$ are possibly high. This helps to mitigate the issues of missing mode and boundary distortion since as in Figure 1, if the target mode 1 can only partly cover the source mode 1, then D_T cannot receive large values from source mode 1. Another important aspect of our approach is to maintain the cluster/manifold structure of source and target data in the joint space via the manifold regularization to avoid the data distortion problem.

3.2 Dual Generator-Discriminator Deep Code Domain Adaptation Network

To address the two inherent problems in the DDAN mentioned in Section 2.4, we employ two different generators G_S and G_T to map source and target domain examples to the joint space and two discriminators D_S and D_T to distinguish source examples against target examples and vice versa together with the source classifier $\mathcal{C}$ which is used to classify the source examples with labels as shown in Figure 2. We name our proposed model as Dual Generator-Discriminator Deep Code Domain Adaptation Network (Dual-GD-DDAN).

Updating the discriminators The two discriminators D_S and D_T are trained to distinguish the source examples against the target examples and vice versa as follows:

$$\min_{D_S}\left(\frac{(1+\theta)}{N_S}\sum_{i=1}^{N_S}\left[-\log D_S\left(G_S\left(\boldsymbol{x}_i^S\right)\right)\right]+\frac{1}{N_T}\sum_{i=1}^{N_T}\left[-\log\left[1-D_S\left(G_T\left(\boldsymbol{x}_i^T\right)\right)\right]\right]\right) \tag{2}$$

$$\min_{D_T}\left(\frac{1}{N_S}\sum_{i=1}^{N_S}\left[-\log\left[1-D_T\left(G_S\left(\boldsymbol{x}_i^S\right)\right)\right]\right]+\frac{(1+\theta)}{N_T}\left[-\sum_{i=1}^{N_T}\log D_T\left(G_T\left(\boldsymbol{x}_i^T\right)\right)\right]\right) \tag{3}$$

where $\theta > 0$. Note that a high value of θ encourages D_s and D_T place higher values on $G_S\left(\boldsymbol{x}^S\right)$ and $G_T\left(\boldsymbol{x}^T\right)$ respectively.

Updating the source classifier The source classifier is employed to classify the source examples with labels as follows:

$$\min_{\mathcal{C}}\frac{1}{N_S}\sum_{i=1}^{N_S}\ell\left(\mathcal{C}\left(G_S\left(\boldsymbol{x}_i^S\right)\right),y_i\right)$$

where ℓ specifies the cross-entropy loss function for the binary classification (e.g., using cross-entropy).

Updating the generators The two generators G_S and G_T are trained to i) maintain the manifold/cluster structures of source and target data in their original spaces to avoid the data distortion problem and ii) move the target samples toward the source samples in the joint space and resolve the missing mode and boundary distortion problems in the joint space.

To maintain the manifold/cluster structures of source and target data in their original spaces, we propose minimizing the manifold regularization term as:

$$\min_{G}\mathcal{M}\left(G_S,G_T\right) \tag{4}$$

where $\mathcal{M}\left(G_S,G_T\right)$ is formulated as:

$$\mathcal{M}\left(G_S,G_T\right)=\sum_{i,j=1}^{N_S}\mu_{ij}\left\|G_S\left(\boldsymbol{x}_i^S\right)-G_S\left(\boldsymbol{x}_j^S\right)\right\|^2+\sum_{i,j=1}^{N_T}\mu_{ij}\left\|G_T\left(\boldsymbol{x}_i^T\right)-G_T\left(\boldsymbol{x}_j^T\right)\right\|^2$$

where the weights are defined as $\mu_{ij}=\exp\left\{-\left\|h\left(\boldsymbol{x}_i\right)-h\left(\boldsymbol{x}_j\right)\right\|^2/\left(2\sigma^2\right)\right\}$ with $h\left(\boldsymbol{x}\right)=$ concat $\left(\overleftarrow{h_L}\left(\boldsymbol{x}\right),\overrightarrow{h_L}\left(\boldsymbol{x}\right)\right)$ where $\overrightarrow{h_L}\left(\boldsymbol{x}\right)$ and $\overleftarrow{h_L}\left(\boldsymbol{x}\right)$ are the last hidden states of the bidirectional RNN with input $\boldsymbol{x}$.

To move the target samples toward the source samples and resolve the missing mode and boundary distortion problems in the joint space, we propose minimizing the following objective function:

$$\min_{D}\mathcal{K}\left(G_S,G_T\right) \tag{5}$$

where $\mathcal{K}\left(G_S,G_T\right)$ is defined as:

$$\mathcal{K}\left(G_S,G_T\right)=\frac{1}{N_S}\sum_{i=1}^{N_S}\left[-\log\left[D_T\left(G_S\left(\boldsymbol{x}_i^S\right)\right)\right]\right]+\frac{1}{N_T}\sum_{i=1}^{N_T}\left[-\log\left[D_S\left(G_T\left(\boldsymbol{x}_i^T\right)\right)\right]\right] \tag{6}$$

Moreover, the source generator G_S has to work out the representation that is suitable for the source classifier, hence we need to minimize the following objective function:

$$\min_{G_S}\frac{1}{N_S}\sum_{i=1}^{N_S}\ell\left(\mathcal{C}\left(G_S\left(\boldsymbol{x}_i^S\right)\right),y_i\right)$$

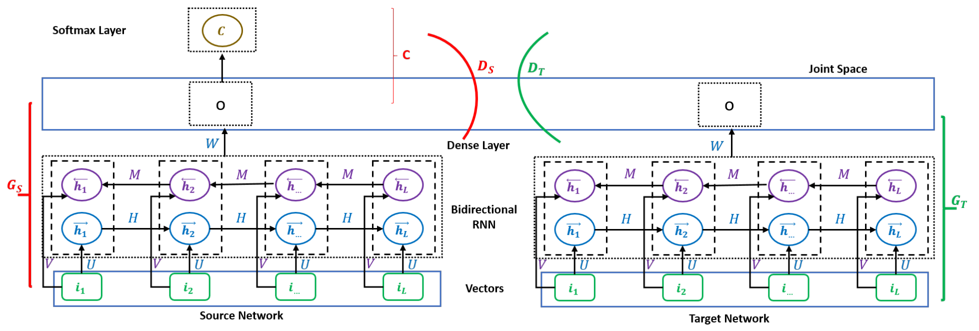


Figure 2: The architecture of our Dual Generator-Discriminator Deep Code Domain Adaptation Network (Dual-GD-DDAN). The generators G_S and G_T take the sequential code tokens of the source domain and target domain in vectorial form respectively and map this sequence to the joint layer (i.e., the joint space). The discriminators D_S and D_T are invoked to discriminate the source and target data. The source classifier $\mathcal{C}$ is trained on the source domain with labels. We note that the source and target networks do not share parameters and are not identical.

Finally, to update G_S and G_T, we need to minimize the following objective function:

$$\frac{1}{N_S}\sum_{i=1}^{N_S}\ell\left(\mathcal{C}\left(G_S\left(\boldsymbol{x}_i^S\right)\right), y_i\right) + \alpha\mathcal{M}\left(G_S, G_T\right) + \beta\mathcal{K}\left(G_S, G_T\right)$$

where α, $\beta > 0$ are two non-negative parameters.

3.3 The rationale for our Dual Generator-Discriminator Deep Code Domain Adaptation Network approach

Below we explain why our proposed Dual-GD-DDAN is able to resolve the two critical problems that occur with the DDAN approach. First, if $\boldsymbol{x}_i^S$ and $\boldsymbol{x}_j^S$ are proximal to each other and are located in the same cluster, then their representations $h\left(\boldsymbol{x}_i^S\right)$ and $h\left(\boldsymbol{x}_j^S\right)$ are close and hence, the weight μ_{ij} is large. This implies $G_S\left(\boldsymbol{x}_i^S\right)$ and $G_S\left(\boldsymbol{x}_j^S\right)$ are encouraged to be close in the joint space because we are minimizing $\mu_{ij}\left\|G_S\left(\boldsymbol{x}_i^S\right) - G_S\left(\boldsymbol{x}_j^S\right)\right\|^2$ as shown in Eq. (4). This increases the chance of the two representations residing in the same cluster in the joint space. Therefore, Dual-GD-DDAN is able to preserve the clustering structure of the source data in the joint space. By using the same argument, we reach the same conclusion for the target domain.

Second, following Eqs. (2, 3), the discriminator D_S is trained to encourage large values for the source modes (i.e., $G_S\left(\boldsymbol{x}^S\right)$), while the discriminator D_T is trained to produce large values for the target modes (i.e., $G_T\left(\boldsymbol{x}^T\right)$). Moreover, as in Eq. (6), G_s is trained to move the source domain examples $\boldsymbol{x}^S$ to the high-valued region of D_T (i.e., the target modes or $G_T\left(\boldsymbol{x}^T\right)$) and G_T is trained to move the target examples $\boldsymbol{x}^T$ to the high-valued region of D_S (i.e., the source modes or $G_S\left(\boldsymbol{x}^S\right)$). As a consequence, eventually, the source modes (i.e., $G_S\left(\boldsymbol{x}^S\right)$) and target modes (i.e., $G_T\left(\boldsymbol{x}^T\right)$) overlap, while D_S and D_T place large values on both source (i.e., $G_S\left(\boldsymbol{x}^S\right)$) and target (i.e., $G_T\left(\boldsymbol{x}^T\right)$) modes. The mode missing problem is less likely to happen since, as shown in Figure 1, if the target data misses source mode 2, then D_T cannot receive large values from source mode 2. Similarly, the boundary distortion problem is also less likely to happen since as in Figure 1, if the target mode 1 can only partly cover the source mode 1, then D_T cannot receive large values from source mode 1. Therefore, Dual-GD-DDAN allows us to reduce the impact of the missing mode and boundary distortion problems, hence making the target distribution more identical to the source distribution in the joint space.

3.4 Dual Generator-Discriminator Semi-supervised Deep Code Domain Adaptation Network

When successfully bridging the gap between the source and target domains in the joint layer (i.e., the joint space), the target samples can be regarded as the unlabeled portion of a semi-supervised learning problem. Based on this observation, Nguyen et al. (Nguyen et al., 2019) proposed to enforce the clustering assumption (Chapelle & Zien, 2005) by minimizing the conditional entropy and using the spectral graph to inspire the smoothness of the source classifier $\mathcal{C}$. Using our proposed Dual-GD-DDAN, the conditional entropy $\mathcal{H}\left(\mathcal{C}, G_S, G_T\right)$ is defined as:

$$\begin{aligned}\mathcal{H}\left(\mathcal{C}, G_S, G_T\right) = \mathbb{E}_{\boldsymbol{x}\sim\mathbb{P}_S}\left[-\mathcal{C}\left(G_S\left(\boldsymbol{x}\right)\right)\log\left[C\left(G_S\left(\boldsymbol{x}\right)\right)\right]\right] + \mathbb{E}_{\boldsymbol{x}\sim\mathbb{P}_S}\left[-\left[1-\mathcal{C}\left(G_S\left(\boldsymbol{x}\right)\right)\right]\log\left[1-\mathcal{C}\left(G_S\left(\boldsymbol{x}\right)\right)\right]\right] \\ + \mathbb{E}_{\boldsymbol{x}\sim\mathbb{P}_T}\left[-\mathcal{C}\left(G_T\left(\boldsymbol{x}\right)\right)\log\left[\mathcal{C}\left(G_T\left(\boldsymbol{x}\right)\right)\right]\right] + \mathbb{E}_{\boldsymbol{x}\sim\mathbb{P}_T}\left[-\left[1-\mathcal{C}\left(G_T\left(\boldsymbol{x}\right)\right)\right]\log\left[1-\mathcal{C}\left(G_T\left(\boldsymbol{x}\right)\right)\right]\right]\end{aligned}$$

Let $SG = \left(\mathcal{V}, \mathcal{E}\right)$ where the set of vertices $\mathcal{V} = S \cup T$ be the spectral graph defined as in (Nguyen et al., 2019). The smoothness-inspired term is defined as:

$$\mathcal{S}\left(\mathcal{C}, G_S, G_T\right) = \sum_{(\boldsymbol{u},\boldsymbol{v})\in\mathcal{E}} \mu_{\boldsymbol{uv}} KL\left(B_{\boldsymbol{u}}, B_{\boldsymbol{v}}\right) = \sum_{(\boldsymbol{u},\boldsymbol{v})\in\mathcal{E}} \mu_{\boldsymbol{uv}}\left[C\left(\boldsymbol{u}\right)\log\frac{C\left(\boldsymbol{u}\right)}{C\left(\boldsymbol{v}\right)} + \left(1-C\left(\boldsymbol{u}\right)\right)\log\frac{1-C\left(\boldsymbol{u}\right)}{1-C\left(\boldsymbol{v}\right)}\right]$$

where $B_{\boldsymbol{u}}$ specifies the Bernoulli distribution with $\mathbb{P}\left(y=1 \mid \boldsymbol{u}\right) = \mathcal{C}\left(\boldsymbol{u}\right)$ and $\mathbb{P}\left(y=-1 \mid \boldsymbol{u}\right) = 1-\mathcal{C}\left(\boldsymbol{u}\right)$. The weight $\mu_{\boldsymbol{uv}} = \exp\left\{-\left\|\boldsymbol{u}-\boldsymbol{v}\right\|^2 / \left(2\sigma^2\right)\right\}$, and $KL\left(B_{\boldsymbol{u}}, B_{\boldsymbol{v}}\right)$ specifies the Kullback-Leibler divergence between two distributions. Here we note that $\boldsymbol{u} = G_S\left(\boldsymbol{x}^S\right)$ and $\boldsymbol{v} = G_T\left(\boldsymbol{x}^T\right)$ are two representations of the source sample $\boldsymbol{x}^S$ and the target sample $\boldsymbol{x}^T$ in the joint space. We incorporate these two terms into our Dual Generator-Discriminator mechanism to propose Dual Generator-Discriminator Semi-supervised Deep Code Domain Adaptation Network (Dual-GD-SDDAN) with the following objective function:

$$\frac{1}{N_S}\sum_{i=1}^{N_S}\ell\left(\mathcal{C}\left(G_S\left(\boldsymbol{x}_i^S\right)\right), y_i\right) + \alpha\mathcal{M}\left(G_S, G_T\right) + \beta\mathcal{K}\left(G_S, G_T\right) + \gamma\mathcal{H}\left(\mathcal{C}, G_S, G_T\right) + \lambda\mathcal{S}\left(\mathcal{C}, G_S, G_T\right)$$

where γ, λ are two non-negative parameters.

4 Experiments

We present experimental results of applying our Dual-GD-DDAN approach to five real-world software projects (Lin et al., 2018). We compare our proposed Dual-GD-DDAN with VulDeePecker without domain adaptation, MMD, D2GAN, DIRT-T and DDAN using the architecture CD$\boldsymbol{A}$N proposed in (Nguyen et al., 2019). We further compare our proposed *Dual Generator-Discriminator Semi-supervised Deep Code Domain Adaptation* (Dual-GD-SDDAN) and *Semi-supervised Deep Code Domain Adaptation* (SCDAN) introduced in (Nguyen et al., 2019).

4.1 Experimental Setup

4.1.1 Experimental Data Set

We use the real-world data sets collected by (Lin et al., 2018), which contain the source code of vulnerable and non-vulnerable functions obtained from five real-world software projects, namely FFmpeg (#vul-funcs: 187, #non-vul-funcs: 5,427), LibTIFF (#vul-funcs: 81, #non-vul-funcs: 695), LibPNG (#vul-funcs: 43, #non-vul-funcs: 551), VLC (#vul-funcs: 25, #non-vul-funcs: 5,548) and Pidgin (#vul-funcs: 42, #non-vul-funcs: 8,268) where #vul-funcs and #non-vul-funcs is the number of vulnerable and non-vulnerable functions respectively. The data sets contain both multimedia (FFmpeg, VLC, Pidgin) and image (LibPNG, LibTIFF) application categories. In our experiment, some of the data sets from the multimedia category were used as the source domain whilst other data sets from the image category were used as the target domain (see Table 1).

4.1.2 Model Configuration

For training the eight methods – VulDeePecker, MMD, D2GAN, DIRT-T, DDAN, Dual-GD-DDAN, SCDAN and Dual-GD-SDDAN – we use one-layer bidirectional recurrent neural networks with LSTM cells where the size of hidden states is in $\{128, 256\}$ for the generators. For the source classifier and discriminators, we use deep feed-forward neural networks with two hidden layers in which 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. We employ the Adam optimizer (Kingma & Ba, 2014) with an initial learning rate in $\{0.001, 0.0001\}$. The mini-batch size is 64. The trade-off parameters α, β, γ, λ are in $\{10^{-1}, 10^{-2}, 10^{-3}\}$, θ is in $\{0, 1\}$ and $1/(2\sigma^2)$ is in $\{2^{-10}, 2^{-9}\}$.

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 VulDeePecker, MMD, D2GAN, DIRT-T, DDAN, Dual-GD-DDAN, SCDAN and Dual-GD-SDDAN without using any label information while the second partition contains 20% for testing the models. We additionally apply gradient clipping regularization to prevent over-fitting in the training process of each model. We implement eight mentioned methods in Python using Tensorflow which is an open-source software library for Machine Intelligence developed by the Google Brain Team. We run our experiments on a computer with an Intel Xeon Processor E5-1660 which had 8 cores at 3.0 GHz and 128 GB of RAM. For each method, we run the experiments 5 times and then record the average predictive performance.

4.2 Experimental Results

4.2.1 Code Domain Adaptation for a Fully Non-labeled Target Project

Quantitative Results We first investigate the performance of our proposed Dual-GD-DDAN compared with other methods including VulDeePecker (VULD) without domain adaptation (Li et al., 2016), DDAN (Nguyen et al., 2019), MMD (Long et al., 2015), D2GAN (Nguyen et al., 2017) and DIRT-T (Shu et al., 2018) with VAP applied in the joint feature layer using the architecture CD*A*N introduced in (Nguyen et al., 2019). The VulDeePecker method is only trained on the source data and then tested on the target data, while the MMD, D2GAN, DIRT-T, DDAN and Dual-GD-DDAN methods employ the target data without using any label information for domain adaptation.

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 and Dual-GD-DDAN for predicting vulnerable and non-vulnerable code functions on the testing set of the target domain (Best performance in **bold**).

Source $\rightarrow$ Target	Methods	FNR	FPR	Recall	Precision	F1-measure
Pidgin $\rightarrow$ 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%
	Dual-GD-DDAN	**33.33%**	**0%**	**66.67%**	**100%**	**80%**
FFmpeg $\rightarrow$ 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%
	Dual-GD-DDAN	**12.5%**	8.2%	**87.5%**	56%	**68.29%**
FFmpeg $\rightarrow$ 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%
	Dual-GD-DDAN	**0%**	**2.17%**	**100%**	**80%**	**88.89%**
VLC$\rightarrow$ 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%
	Dual-GD-DDAN	**28.57%**	2.15%	**71.43%**	71.43%	**71.43%**
Pidgin $\rightarrow$ 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%
	Dual-GD-DDAN	29.41%	**6.76%**	70.59%	57.14%	**63.16%**

In Table 1, the experimental results *show that our proposed Dual-GD-DDAN achieves a higher performance for detecting vulnerable and non-vulnerable functions for most performance measures*, including FNR, FPR, Recall, Precision and F1-measure in almost cases of the source and target domains, especially for F1-measure. Particularly, our Dual-GD-DDAN always obtains the highest F1-measure in all cases. For example, for the case of the source domain (FFmpeg) and target domain (LibPNG), Dual-GD-DDAN achieves an F1-measure of 88.89% compared with an F1-measure of 84.21%, 84.21%, 80%, 77.78% and 75% obtained with DDAN, DIRT-T, D2GAN, MMD and VulDeePecker respectively.

Table 2: Accuracies obtained by the DDAN and Dual-GD-DDAN methods when predicting vulnerable and non-vulnerable code functions on the source and target domains. Note that tr src, ts tar, tr tar, ts src, and acc gap are the shorthands of train source, test target, train target, test source, and accuracy gap respectively. For the accuracy gap, a smaller value is better.

Source → Target	**Methods**	**Accuracy**			**Accuracy**		
		Tr src / Ts tar/acc gap			**Tr tar / Ts src/ acc gap**		
Pidgin → LibPNG	DDAN	98.8%	96%	2.8%	97%	92%	5%
	Dual-GD-DDAN	99%	97%	**2%**	97%	95%	**2%**
FFmpeg → LibPNG	**Methods**	**Accuracy**			**Accuracy**		
		Tr src / Ts tar/acc gap			**Tr tar / Te src/acc gap**		
	DDAN	95.9%	92%	3.9%	91%	83.3%	7.7%
	Dual-GD-DDAN	97%	96%	**1%**	98%	95.6%	**2.4%**

4.2.2 Boundary Distortion Analysis

Quantitative Results To quantitatively demonstrate the efficiency of our proposed Dual-GD-DDAN in alleviating the boundary distortion problem caused by using the GAN principle, we reuse the experimental setting in Section 5.2 (Santurkar et al., 2018). The basic idea is, given two data sets S_1 and S_2, to quantify the degree of cover of these two data sets. We train a classifier $\mathcal{C}_1$ on S_1, then test on S_2 and another classifier $\mathcal{C}_2$ on S_2, then test on S_1. If these two data sets cover each other well with reduced boundary distortion, we expect that if $\mathcal{C}_1$ predicts well on S_1, then it should predict well on S_2 and vice versa if $\mathcal{C}_2$ predicts well on S_2, then it should predict well on S_1. This would seem reasonable since if boundary distortion occurs (i.e., assume that S_2 partly covers S_1), then $\mathcal{C}_2$ trained on S_2 would struggle to predict S_1 well which is much larger and possibly more complex. Therefore, we can utilize the magnitude of the accuracies and the accuracy gap of $\mathcal{C}_1$ and $\mathcal{C}_2$ when predicting their training and testing sets to assess the severity of the boundary distortion problem.

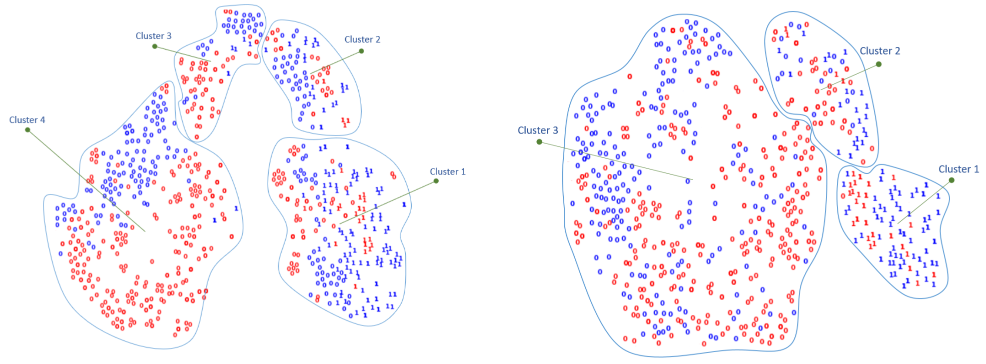


Figure 3: A 2D t-SNE projection for the case of the FFmpeg → LibPNG domain adaptation. The blue and red points represent the source and target domains in the joint space respectively. In both cases of the source and target domains, data points labeled 0 stand for non-vulnerable samples and data points labeled 1 stand for vulnerable samples.

Inspired by this observation, we compare our Dual-GD-DDAN with DDAN using the representations of the source and target samples in the joint feature space corresponding to their best models. In particular, for a given pair of source and target data sets and for comparing each method, we train a neural network classifier on the best representations of the source data set in the joint space, then predict on the source and target data set and do the same but swap the role of the source and target data sets. We then measure the difference of the corresponding accuracies as a means of measuring the severity of the boundary distortion. We choose to conduct such a boundary distortion analysis for two pairs of the source (FFmpeg and Pidgin) and target (LibPNG) domains. As shown in Table 2, *all gaps obtained by our Dual-GD-DDAN are always smaller than those obtained by DDAN*, while the accuracies obtained by our proposed method are always larger. We can therefore conclude that our Dual-GD-DDAN method produces a better representation for source and target samples in the joint space and is less susceptible to boundary distortion compared with the DDAN method.

Visualization We further demonstrate the efficiency of our proposed Dual-GD-DDAN in alleviating the boundary distortion problem caused by using the GAN principle. Using a t-SNE (Laurens & Geoffrey, 2008) projection, with perplexity equal to 30, we visualize the feature distributions of the source and target domains in the joint space. Specifically, we project the source and target data in the joint space (i.e., $G(\boldsymbol{x})$) into a 2D space with domain adaptation (DDAN) and with dual-domain adaptation (Dual-GD-DDAN). In Figure 3, we observe these cases when performing domain adaptation from a software project (FFmpeg) to another (LibPNG). As shown in Figure 3, with undertaking domain adaptation (DDAN, the left figure) and dual-domain adaptation (Dual-GD-DDAN, the right figure), *the source and target data sampled are intermingled especially for Dual-GD-DDAN*. However, it can be observed that DDAN when solely applying the GAN is seriously vulnerable to the boundary distortion issue. In particular, in the clusters/data modes 2, 3 and 4 (the left figure), the boundary distortion issue occurs since the blue data only partly cover the corresponding red ones (i.e., the source and target data do not totally mix up). Meanwhile, for our Dual-GD-DDAN, *the boundary distortion issue is much less vulnerable*, and the mixing-up level of source and target data is significantly higher in each cluster/data mode.

4.2.3 Quantitative results of Dual Generator-Discriminator Semi-supervised Deep Code Domain Adaptation

In this section, we compare the performance of our *Dual Generator-Discriminator Semi-supervised Deep Code Domain Adaptation* (Dual-GD-SDDAN) with *Semi-supervised Deep Code Domain Adaptation* (SCDAN) (Nguyen et al., 2019) on four pairs of source and target domain including FFmpeg → LibTIFF, FFmpeg → LibPNG, VLC→ LibPNG and Pidgin → LibTIFF. In Table 3, *the experimental results show that our Dual-GD-SDDAN achieves a higher performance than SCDAN for detecting vulnerable and non-vulnerable functions* in terms of FPR, Precision and F1-measure in almost cases of the source and target domains, especially for F1-measure. For example, to the case of the source domain (VLC) and target domain (LibPNG), our Dual-GD-SDDAN achieves an F1-measure of 76.19% compared with an F1-measure of 72.73% obtained with SCDAN. These results further demonstrate the ability of our Dual-GD-SDDAN for dealing with the mode collapsing problem better than SCDAN (Nguyen et al., 2019), hence obtaining better predictive performance in the context of software domain adaptation.

Table 3: Performance results in terms of false negative rate (FNR), false positive rate (FPR), Recall, Precision and F1-measure of SCDAN and Dual-GD-SDDAN for predicting vulnerable and non-vulnerable code functions on the testing set of the target domain (Best performance in **bold**).

Source → Target	Methods	FPR	FNR	Recall	Precision	F1-measure
FFmpeg → LibTIFF	SCDAN	5.38%	**14.29%**	**85.71%**	57.14%	68.57%
	Dual-GD-SDDAN	**3.01%**	35.29%	64.71%	**73.33%**	**68.75%**
FFmpeg → LibPNG	SCDAN	1.08%	**12.5%**	**87.5%**	87.5%	87.5%
	Dual-GD-SDDAN	**0%**	17.5%	82.5%	**100%**	**90.41%**
VLC→ LibPNG	SCDAN	**1.06%**	33.33%	66.67%	**80%**	72.73%
	Dual-GD-SDDAN	4.39%	**11.11%**	**88.89%**	66.67%	**76.19%**
Pidgin → LibTIFF	SCDAN	5.56%	**30%**	**70%**	58.33%	63.64%
	Dual-GD-SDDAN	**2.98%**	37.5%	62.5%	**71.43%**	**66.67%**

5 Conclusion

Software vulnerability detection (SVD) is an important problem in the software industry and in the field of computer security. One of the most crucial issues in SVD is to cope with the scarcity of labeled vulnerabilities in projects that require the laborious labeling of code by software security experts. In this paper, we propose the Dual Generator-Discriminator Deep Code Domain Adaptation Network (Dual-GD-DDAN) method to deal with the missing mode and boundary distortion problems which arise from the use of the GAN principle when reducing the discrepancy between source and target data in the joint space. We conducted experiments to compare our Dual-GD-DDAN method with the state-of-the-art baselines. The experimental results show that our proposed method outperforms these rival baselines by a wide margin in term of predictive performances.

References

M. Almorsy, J.C. Grundy, and A. Ibrahim. Supporting automated vulnerability analysis using formalized vulnerability signatures. In *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, ASE 2012, pp. 100–109. ACM, 2012. ISBN 978-1-4503-1204-2. doi: 10.1145/2351676.2351691. URL `http://doi.acm.org/10.1145/2351676.2351691`.

K. M. Borgwardt, A. Gretton, M. J. Rasch, H-P Kriegel, B. Schölkopf, and A. J. Smola. Integrating structured biological data by kernel maximum mean discrepancy. *Bioinformatics*, 22(14):e49–e57, 2006. ISSN 1367-4803.

L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

O. Chapelle and A. Zien. Semi-supervised classification by low density separation. In *AISTATS*, pp. 57–64. Citeseer, 2005.

J. Chung, K. Kastner, L. Dinh, K. Goel, A. C. Courville, and Y. Bengio. A recurrent latent variable model for sequential data. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 28*, pp. 2980–2988. Curran Associates, Inc., 2015.

C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

N. Courty, R. Flamary, D. Tuia, and A. Rakotomamonjy. Optimal transport for domain adaptation. *IEEE transactions on pattern analysis and machine intelligence*, 39(9):1853–1865, 2017.

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.

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, pp. 248–255, 2009.

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

G. French, M. Mackiewicz, and M. Fisher. Self-ensembling for visual domain adaptation. In *International Conference on Learning Representations*, 2018.

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.

Ian Goodfellow. Nips 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160*, 2016.

R. Gopalan, L. Ruonan, and R. Chellappa. Domain adaptation for object recognition: An unsupervised approach. In *Proceedings of the 2011 International Conference on Computer Vision*, ICCV '11, pp. 999–1006, 2011. ISBN 978-1-4577-1101-5.

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.

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. URL `http://arxiv.org/abs/1412.6980`.

A. Kumar, S. Avishek, and D. Hal. Co-regularization based semi-supervised domain adaptation. In J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta (eds.), *Advances in Neural Information Processing Systems 23*, pp. 478–486. 2010.

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

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, 2018.

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.

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.

T. Miyato, S. Maeda, S. Ishii, and M. Koyama. Virtual adversarial training: A regularization method for supervised and semi-supervised learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2018.

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. 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.

S. Purushotham, W. Carvalho, T. Nilanon, and Y. Liu. Variational recurrent adversarial deep domain adaptation. In *International Conference on Learning Representations (ICLR)*, 2017.

Shibani Santurkar, Ludwig Schmidt, and Aleksander Madry. A classification-based study of covariate shift in GAN distributions. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 4480–4489. PMLR, 10–15 Jul 2018.

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.

E. Tzeng, J. Hoffman, T. Darrell, and K. Saenko. Simultaneous deep transfer across domains and tasks. *CoRR*, 2015.

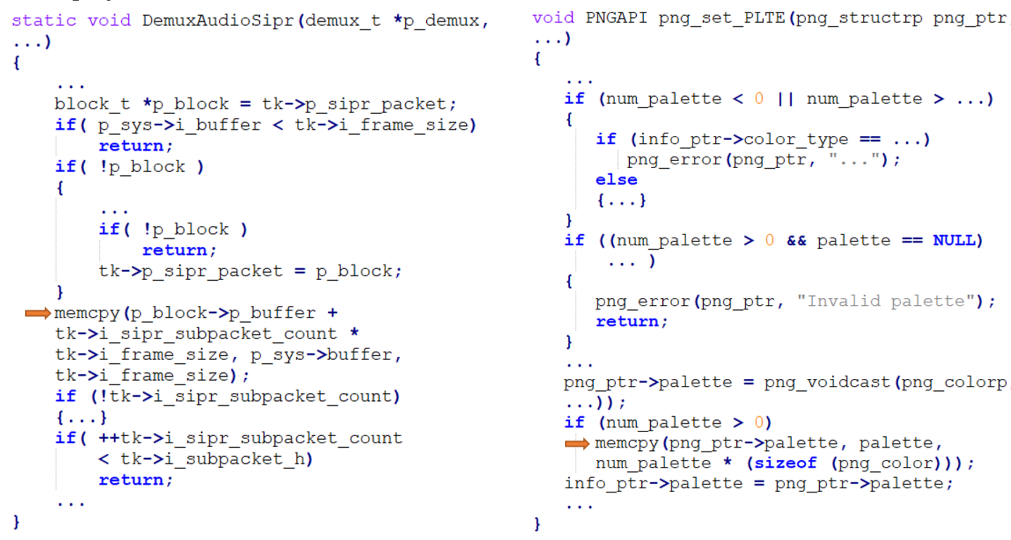
F. Yamaguchi, F. Lindner, and K. Rieck. Vulnerability extrapolation: assisted discovery of vulnerabilities using machine learning. In *Proceedings of the 5th USENIX conference on Offensive technologies*, pp. 13–23, 2011.

T. Yao, Yingwei Pan, C. Ngo, Houqiang Li, and Tao Mei. Semi-supervised domain adaptation with subspace learning for visual recognition. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 00, pp. 2142–2150, June 2015.

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 Motivating Example

We give an example of source code functions obtained from the VLC and LibPNG projects, to demonstrate that transfer learning for software vulnerability detection between different projects is plausible and promising. Both C language functions obtained from the VLC and LibPNG projects depicted in Figure 4 invoke the *memcpy* function which is used to copy one memory buffer to another. The misuse of this function can cause a buffer overflow error if insufficient memory is allocated in the target buffer for all of the data to be copied from the source buffer. Furthermore, these functions also share rather similar semantic and syntactic relationships (i.e. the C language programming syntax, loop structure etc). Therefore, a model that can capture the characteristics of the first function in the first project should be able to confidently predict the second function in the second project. It therefore makes sense to undertake transfer learning from the first project to the second project.



```c
static void DemuxAudioSipr(demux_t *p_demux,
...)
{
    ...
    block_t *p_block = tk->p_sipr_packet;
    if( p_sys->i_buffer < tk->i_frame_size)
        return;
    if( !p_block )
    {
        ...
        if( !p_block )
            return;
        tk->p_sipr_packet = p_block;
    }
    memcpy(p_block->p_buffer +
    tk->i_sipr_subpacket_count *
    tk->i_frame_size, p_sys->buffer,
    tk->i_frame_size);
    if (!tk->i_sipr_subpacket_count)
    {...}
    if( ++tk->i_sipr_subpacket_count
        < tk->i_subpacket_h)
        return;
    ...
}
```

```c
void PNGAPI png_set_PLTE(png_structrp png_ptr,
...)
{
   ...
   if (num_palette < 0 || num_palette > ...)
   {
      if (info_ptr->color_type == ...)
         png_error(png_ptr, "...");
      else
      {...}
   }
   if ((num_palette > 0 && palette == NULL)
       ... )
   {
      png_error(png_ptr, "Invalid palette");
      return;
   }
   ...
   png_ptr->palette = png_voidcast(png_colorp,
   ...));
   if (num_palette > 0)
      memcpy(png_ptr->palette, palette,
      num_palette * (sizeof (png_color)));
   info_ptr->palette = png_ptr->palette;
   ...
}
```

Figure 4: An example of two source code functions (with some parts omitted for brevity) in the C programming language obtained from the VLC (Left) and LibPNG project (Right). These two source code examples highlight the same vulnerability due to the misuse of the *memcpy* function.

B Related Work

In this section, we introduce work related to ours. First, we present the recent work in automatic feature learning for software vulnerability detection. Finally, we present the recent work in deep domain adaptation.

Automatic feature learning in software vulnerability detection minimizes intervention from security experts (Li et al., 2018; Lin et al., 2018; Dam et al., 2018). Particularly, (Dam et al., 2018; Lin et al., 2018) shared the same approach employing a Recurrent Neutral Network (RNN) to transform sequences of code tokens to vectorial features for automatic feature learning, which are then fed to a separate classifier (e.g., Support Vector Machine (Cortes & Vapnik, 1995) or Random Forest (Breiman, 2001)) for classification purposes. However, owing to the independence of learning the vector representations and training the classifier, it is likely that the resulting vector representations of (Lin et al., 2018; Dam et al., 2018) may not fit well with classifiers to enhance the predictive performance. To deal with this problem, the study introduced in (Li et al., 2018) combined the learning of the vector representations and the training of a classifier in a deep neural network. This work leverages a bidirectional RNN to take sequential data as inputs and the outputs from the bidirectional RNN are then fed to a deep feed-forward neural network for prediction.

Deep domain adaptation aims to bridge the gap between the source and target domains in a joint space (Ganin & Lempitsky, 2015; Tzeng et al., 2015; Shu et al., 2018; French et al., 2018). These methods try to minimize a divergence (e.g., Jensen-Shannon divergence, f-divergence, maximum mean discrepancy (MMD), or Wasserstein distance) between the source and target distributions in the joint space. For instance, (Ganin & Lempitsky, 2015; Tzeng et al., 2015; Long et al., 2015; Shu et al., 2018; French et al., 2018) minimize the Jensen-Shannon divergence between two relevant distributions relying on the GAN principle (Goodfellow et al., 2014), while (Long et al., 2015) minimizes the MMD and the work of (Courty et al., 2017) minimizes the Wasserstein distance between two relevant distributions. The study proposed in (Nguyen et al., 2017) relies on the GAN principle with using two discriminators aiming to tackle the problem of mode collapse encountered in generative adversarial networks (GANs). In addition, most of aforementioned works proposed to transfer a pretrained model on the data set ImageNet (Deng et al., 2009) to other image sources. The work of (Purushotham et al., 2017) proposed to apply deep domain adaptation for multivariate time-series data. This work based on the Variational RNN (Chung et al., 2015) and the GAN principle was applied on the latent representation. Recently, relying on the GAN principle (Goodfellow et al., 2014), the work of (Nguyen et al., 2019) proposed to tackle sequential inputs, particularly source code, in software domain adaptation. The underlying idea is to use the GAN principle to close the gap between the source and target domains in the joint space and enforce the clustering assumption to utilize the information of unlabeled target samples in a semi-supervised learning context.

Leveraging semi-supervised learning with domain adaptation has been studied in shallow domain adaptation (Kumar et al., 2010; Yao et al., 2015). DIRT-T (Shu et al., 2018) leveraged semi-supervised context by enforcing the clustering assumption via minimizing the conditional entropy and using virtual adversarial perturbation (VAP) (Miyato et al., 2018) to impose by smoothness over the decision output. Another work of (Nguyen et al., 2019) proposed to leverage semi-supervised context by enforcing the clustering assumption via minimizing the conditional entropy and manifold regularization with spectral graph which was proven to be more appropriate to discrete sequential data like source codes.