electronics

MDPI

Article

Developing Cross-Domain Host-Based Intrusion Detection

Oluwagbemiga Ajayi [1,2,*], Aryya Gangopadhyay [1,2], Robert F. Erbacher [3] and Carl Bursat [3]

[1] Department of Information Systems, University of Maryland Baltimore County, Baltimore, MD 21250, USA
[2] UMBC Center for Real-Time Distributed Sensing and Autonomy, Catonsville, MD 21228, USA
[3] DEVCOM Army Research Laboratory, Adelphi, MD 20783, USA
* Correspondence: dk60890@umbc.edu

Abstract: Digital transformation has continued to have a remarkable impact on industries, creating new possibilities and improving the performance of existing ones. Recently, we have seen more deployments of cyber-physical systems and the Internet of Things (IoT) as in no other time. However, cybersecurity is often an afterthought in the design and implementation of many systems; therefore, there usually is an introduction of new attack surfaces as new systems and applications are being deployed. Machine learning has been helpful in creating intrusion detection models, but it is impractical to create attack detection models with acceptable performance for every single computing infrastructure and the various attack scenarios due to the cost of collecting quality labeled data and training models. Hence, there is a need to develop models that can take advantage of knowledge available in a high resource source domain to improve performance of a low resource target domain model. In this work, we propose a novel cross-domain deep learning-based approach for attack detection in Host-based Intrusion Detection Systems (HIDS). Specifically, we developed a method for candidate source domain selection from among a group of potential source domains by computing the similarity score a target domain records when paired with a potential source domain. Then, using different word embedding space combination techniques and transfer learning approach, we leverage the knowledge from a well performing source domain model to improve the performance of a similar model in the target domain. To evaluate our proposed approach, we used Leipzig Intrusion Detection Dataset (LID-DS), a HIDS dataset recorded on a modern operating system that consists of different attack scenarios. Our proposed cross-domain approach recorded significant improvement in the target domains when compared with the results from in-domain approach experiments. Based on the result, the F2-score of the target domain CWE-307 improved from 80% in the in-domain approach to 87% in the cross-domain approach while the target domain CVE-2014-0160 improved from 13% to 85%.

Keywords: deep learning; cybersecurity; HIDS; transfer learning; word embedding; similarity measure

Citation: Ajayi, O.; Gangopadhyay, A.; Erbacher, R.F.; Bursat, C. Developing Cross-Domain Host-Based Intrusion Detection. *Electronics* **2022**, *11*, 3631. https://doi.org/10.3390/electronics11213631

Academic Editors: Celestine Iwendi, Thippa Reddy Gadekallu and Taeshik Shon

Received: 26 September 2022
Accepted: 31 October 2022
Published: 7 November 2022

1. Introduction

The advances in digital transformation are unlocking new capabilities and improving existing ones across industries. In the automotive industry for example, connected travelers and autonomous driving are going to be central to the future of the industry [1]. Recently, the US Army laid out their vision of an Internet of Battlefield Things (IoBT) with a goal to provide battlefield situational awareness through a network of interconnected sensors, actuators and analytical devices [2]. In a similar development, the Army Research Lab in 2020 commenced the Artificial Intelligence (AI) for Command and Control (C2) of Multi-Domain Operations (MDO) project with the goal of improving the execution and decision making speed of MDOs by leveraging AI techniques which are lately known for performing better than humans cognitive ability [3]. Meanwhile, in the healthcare industry, smart care, care anywhere, empowered care, and intelligent healthcare are some of the digital transformation ideas with potentials to change how healthcare is delivered going

forward [1]. The obvious outcome of these is reflected through various metrics, including how customers are more conveniently able to access services and the improvement of organizations' bottom-line. Unfortunately, these positive developments are not without their unintended negative consequences, such as information security compromises.

In 2014, the Google Security team made a late discovery of the Heartbleed attack; an attack believed to have affected about 17% of all the websites and arguably the biggest attack in the history of cyber-attacks [4]. Yahoo reacted to a famous 2013 data breach attack with a statement that admitted that hackers had gained access to data of at least 500 million users and therefore encouraged all users who have not changed their passwords since 2014 to please do so [4]. Later recalculation of the impact of this attack revealed that about three billion user accounts were affected [5]. Recently, SolarWinds was in the news as the subject of a cyber-attack that went undetected for several months. US government departments (including Homeland Security and Treasury Department) and some private companies were the main targets [6].

Just as the frequency of cyber-attack incidents has increased, the cost has also continued to rise over the years. Within the last ten years, cyber-attack incidents recording financial losses in excess of $1 Million increased from 21 in 2009 to 105 in 2019 [7]. In their 2021 edition of the Cost of Data Breach Report, IBM and Ponemon Institute reported a data breach cost of $4.24 Million in 2021, which represents a 10% increase from the previous year [8]. Cybersecurity Ventures in their research report [9] stated that the global annual cost of cybercrime damages as of 2015 was $3 trillion with a projected 15% annual increase. By 2025, the cost is expected to be more than $10 trillion.

Over the years, a lot of research efforts have gone into intrusion prevention and detection with some notable improvements but the growth of digital transformation has continued to unfortunately arrive with new attack surfaces that hackers are waiting to uncover and exploit. Hackers are able to wreak much havoc and make organizations pay this heavy price, partly because some cyber-attack incidents stay undetected for a long time after the initial intrusion. Attacks at such stages are called zero-day attacks because their traces present patterns of interest that does not exactly match existing patterns in the detection system [10].

Based on source of origin, attacks can be categorized into insider and outsider attack types. A Firewall is known to provide some degree of protection against outsider attacks but is powerless against insider attacks [11]. Both Intrusion Prevention Systems (IPS) and Intrusion Detection Systems (IDS) are helpful against insider attack types. According to the definition in [12], "An IDS is a type of security tool that monitors network traffic and scans the system for suspicious activities and alerts the system or network administrator", while IPS as described in [13] is a system that can detect intrusion attempts and autonomously interrupt the attempts, thereby preventing the attack. IDSs are categorized mainly into Host-based Intrusion Detection Systems (HIDS) and Network Intrusion Detection (NIDS). HIDS monitors individual hosts and alerts the user should a suspicious activity be detected while NIDS on the other hand are situated at network points to detect intrusions in the network traffic [12,14]. Early efforts to categorize types of HIDS identified Audit data OS-level HIDS and Audit data Application-level HIDS [15,16]. Audit data OS-level HIDS leverages system calls, file system modifications and user logons in building a detection engine.

Traditional Machine Learning techniques such as Artificial Neural Networks (ANN), Decision Trees (DT), and Support Vector Machines (SVM) have been quite instrumental in developing the intrusive behavior and attack detection process [12,14,17] but recording good performance from these algorithms requires good feature engineering and a good quantity of high quality data. It has also been observed that there is a limit to benefits traditional machine learning algorithms could derive from large datasets [18]. Conversely, deep learning approaches are popular for automatic feature extraction and their tendency to improve a lot as data size increases, hence they have been gaining a lot of attention with classification tasks such as intrusion detection model building [19,20]. However, both traditional machine learning and deep learning algorithms taking the in-domain

approach make an underlying assumption that training data (source domain) and test data (target domain) belong to the same feature space and share the same distribution [21]. This assumption hardly holds in many real-life setups and becomes even more complicated in multi-class situations. These challenges could regularly lead to degraded performance of machine learning and deep learning algorithms in solving such problems. Therefore, whenever we do not have enough data to train models for all individual attack domains and/or we have varying feature distribution between attack domains, there is a need to find other ways of achieving good performance in straggling target domains.

Transfer learning in machine learning is a method that aims to reuse the knowledge obtained between task domain models [22]. The hope of a successful knowledge transfer rests partly on degree of similarity that exists between domains. Ref. [23] observed in their work that semantic similarity of domains in Natural Language Processing NLP largely determines whether a neural network is transferable. In [24], a knowledge transfer approach, where models are trained on all the source domains and parameters are tuned on data for the target domain, is called cross-domain.

In this paper, we propose a cross-domain approach to the host-based intrusion detection problem. As opposed to the in-domain approach, where classification models are trained directly using labeled data from the target domain [25], a cross-domain approach leverages the knowledge available in a source domain classification model for the benefit of improving performance of a target domain. Our primary goal is the performance improvement of a straggling target domain. We developed a domain selection framework that computed similarity score across domains in order to identify a candidate source domain. We implemented different methods of absorbing knowledge from the source domain before using the fine-tuning strategy of transfer learning to retrain the target domain. Specifically, we made the following contributions:

- Design and development of a method for HIDS domain similarity score measurement
- Design and development of an approach for selection of candidate source domain to work with a given HIDS target domain
- Implementation of new cross-domain approaches for Host-Based Intrusion Detection.
- Alleviating the pains of zero-day attacks against host infrastructures.

The rest of the paper is organized as follows: Section 2 illustrates the background ideas of intrusion detection. In Section 3, we discuss the related work in this area. Section 4 highlights different aspects of our methodology. In Section 5, we present and explain the experimental results. Finally in Section 6 we focus on a conclusion and future directions.

2. Background

This section presents classification of intrusion detection systems in terms of detection approach, data sources and existing public benchmark datasets.

2.1. Intrusion Detection Approaches

From the detection approach standpoint, Intrusion Detection Systems can be mainly categorized into Signature-based and Anomaly-based Intrusion . In [26], a hybrid approach, which is the combination of Signature-based and Anomaly-based approaches, was also identified.

2.1.1. Signature-Based Detection

Signature-based Detection approaches (sometimes called Misuse-based Detection approaches) rely on signatures of known attacks for their detection. As seen on the right hand side of Figure 1, the test samples flow through the signature decision engine for the detection of a possible match. They tend to perform well against attacks already captured in their signatures [27], with few to no false alarms. To ensure they continue to perform well, the database of the signatures has to be updated with new signatures manually. Because of this, they are powerless against attacks whose signatures are not in the database. Seminal works on the application of Signature-based Detection approach with supervised machine learning algorithm includes [28], where Artificial Neural Networks was used to detect

misuse using a data generated by a RealSecure network monitor, [29], where Bayesian Networks was used to detect attack signature [30], where Snort's [31] decision engine was replaced by Decision Tree [32], where Random Forest was used for misuse detection on KDD 1999 data set [33], and [17] where Radial Basis Function RBF Kernel of Support Vector Machines SVM was used on KDD 1999 data set.

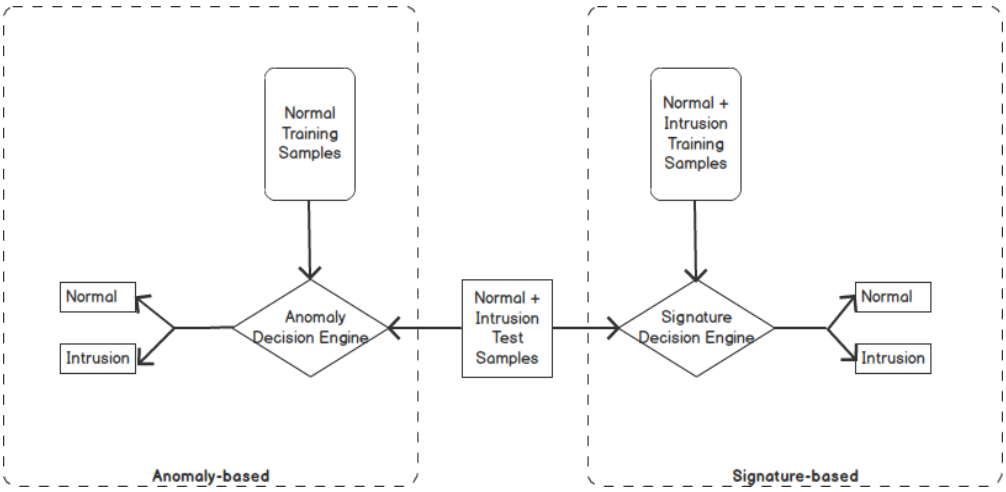


Figure 1. Conceptual working of intrusion detection approaches.

2.1.2. Anomaly-Based Detection

Anomaly-based Detection models have a benign network and system behaviors classifies deviations from these as anomalies. The left hand side of Figure 1 shows that the anomaly detection engine is trained on only normal samples. The main advantage of this approach lies in their ability to detect zero-day attacks. They are however prone to false alarms because unknown network and system behaviors could easily be classified as anomalies. Most of the seminal work on applications of the Anomaly-based Detection approach actually took a hybrid approach [26]; examples will be mentioned after discussing the Hybrid Detection approach.

2.1.3. Hybrid Detection

The Hybrid Detection approach combines the strength of the Signature-based Detection approach and Anomaly-based Detection approach, such that detection rate improves with reduced false positive rates. Seminal works that used this approach using supervised machine learning include [34] that used Bayesian Network to classify events during OS calls with DARPA 1999 dataset, ref. [35] that used Support Vector Machines on simulated and real world Netflow data collected using Flame tool [36], and [37] that used two Random Forest classifiers for signature-based and anomaly-based detection such that results from anomaly-based step formed part of input to the signature-based step.

2.2. Intrusion Detection Data Sources

Another way to categorize Intrusion Detection Systems is through the target infrastructure of attacks they try to detect. From this standpoint we have two categories of IDSs: Network Intrusion Detection Systems (NIDS) and Host-based Intrusion Detection Systems (HIDS). NIDS deployed at different locations within a network, in addition to HIDS and firewalls, can provide a robust protection against attacks [14].

2.2.1. Network Intrusion Detection Systems

NIDS is situated at network points to identify intrusions in the network traffic [12,14], by monitoring packet capture, Netflow and other network sources of data. One of the advantages of NIDS is that external malicious events within the network could be detected

and dealt with before such threats spread to another system [14]. However, NIDS might struggle with high bandwidth networks containing huge volumes of data.

2.2.2. Host-Based Intrusion Detection Systems

HIDS monitors individual hosts by inspecting data of activities from the system and audit sources, and alerts the user should suspicious activities be detected [14]. HIDS is built to be implemented on a single system such that malicious attacks or intrusion that could harm its operating system or data can be detected. Ref. [16] recognized different types of HIDS: Audit data OS-level HIDS and Audit data Application-level HIDS. Audit data OS-level HIDS relates to system calls, file system modifications and user logons.

2.3. IDS Datasets

Cyber-threats have continued to evolve, leading to the changes in the entire cyber-security landscape including relevance of benchmark datasets. For instance, KDD cup (Knowledge Discovery in Database) and DARPA (Defense Advanced Research Projects Agency) datasets are considered outdated [38]. Real IDS datasets are either private or de-identified because they may contain sensitive and private information [13], so there are not too many public datasets. This is especially true of HIDS with only six public system call based HIDS datasets. Details of the public system call-based HIDS datasets are presented in Table 1.

Table 1. List of HIDS datasets.

Year Created	Dataset	Operating System	Provider
1999	KDD	Solaris BSM	Massachusetts Institute of Technology
2006	UNM	SunOS	University of New Mexico
2013	ADFA-LD	Linux	University of New South Wales
2014	ADFA-WD	Windows	University of New South Wales
2017	NGIDS-DS	Linux	University of New South Wales
2019	LID-DS	Linux	Leipzig University

3. Related Work

This section presents a review of existing works in the area of intrusion detection using machine learning approaches. We first discuss works that took the in-domain approach in Sections 3.1 and 3.2. We then discuss works that took the transfer learning approach.

3.1. In-Domain Approaches for Intrusion Detection

The in-domain supervised machine learning approach recorded significant success in the detection of intrusions in both host-based and network intrusion detection systems development. In their work on host-based intrusion detection, Simon et al. [39] compared the performance of Support Vector Machines SVM and Random Forest algorithms on Windows XP HIDS datasets. Representation of features was performed using Distinct Dynamic Link Library Count DDLLC, an approach that essentially transformed the trace sequences to bag of words. To optimize parameters for the machine learning algorithms, they used 5-fold cross validation and grid search. In the result of their experiment on both ADFA-WD and ADFA-WD:SAA, they reported a detection rate of 82% with Random Forest (RF) while SVM recorded 68% for the Radial Basis Function (RBF) kernel and 71% for the sigmoid kernel.

Vinayakumar et al. [40] in their comprehensive exploration of deep learning algorithms for building flexible and effective IDS, evaluated both traditional machine learning algorithm and deep learning algorithms on various datasets of HIDS and NIDS. The section of their work that focused on HIDS datasets used n-gram, bag of words and keras embedding for feature representation and compared the performance of detection models built using SVM and Deep Neural Networks (DNN) algorithms. Experiments on recent

datasets ADFA-LD (a Linux HIDS dataset) and ADFA-WD recorded the best result with DNN of 5 layers and keras embedding feature representation. They reported an accuracy of 83.4% for ADFA-WD and 92.1% for ADFA-LD.

In a comparative study of machine learning algorithms on LID-DS dataset [41], the impact of considering sequences in building host-based intrusion detection was evaluated. For sequence consideration, they selected algorithms in the Recurrent Neural Networks (RNN) family, including RNN and Long Short-term Memory (LSTM), while NB, DT, LR and MLP (all traditional machine learning algorithms) were selected as those not considering sequences. The performance of binary classification models built using these algorithms were compared and results show that RNN and LSTM with accuracies of 88% and 91% outperformed all the other algorithms. Among those not considering sequence, NB recorded the best accuracy at 82%.

In their experiment on NIDS datasets, ref. [40] featured many recent datasets, including UNSW-NB15, CICIDS2017, WSN-DS and Kyoto. Binary and multi-class models were built using algorithms including Logistic Regression (LR), Naive Bayes (NB), KNN, DT, RF, SVM, and DNN (1 to 5 layers). It can be seen through the binary classification results that while DNN achieved better results in few of the experiments, traditional machine learning algorithms are still very competitive for most of the NIDS datasets. For example, LR and SVM each recorded an accuracy of 89.5% while the best performance of DNN was 88.5%. In the multi-class experiments however, RF maintained its good performance while other traditional machine learning algorithms struggled. The strength of DNN became more apparent but RF remained competitive. For example, RF recorded an accuracy of 94.4% on the CICIDS2017 dataset while all DNN of all layer combination outperformed RF with DNN (3 layers) being the best at 96.2%.

While most of these results show good accuracy scores on binary and multi-class models evaluation, plotting the confusion matrix to see the actual performance of individual attack class from experience exposes huge misclassification on some of the attack types.

3.2. Transfer Learning Approaches for Intrusion Detection

It is common among transfer learning-based NIDS works to convert NIDS datasets into images and treat the problem as a computer vision one. Xu et al. [42] designated KDD Cup 99 dataset as the source domain and "corrected" KDD Cup 99 dataset (with 17 intrusion types not found in the source domain) as the target. In the preprocessing step, they transformed the 119 features of the datasets into an 11×11 pixel grayscale image. For the initial training, a deep learning model was built on the source domain data using the Convolutional Neural Networks (CNN) algorithm. Their result shows that CNN recorded an accuracy of 97.9%, outperforming other methods such as SVM, DT, K-nearest Neighbor KNN, and LSTM. Fine-tuning strategy of transfer learning was implemented to adapt to the target domain. Their result shows that post-fine-tuning test performance improved. However, the choice of an outdated benchmark dataset that has been criticized for its many faults [43] is a clear limitation of this work.

Another work by Gangopadhyay et al. [21] took the approach of converting datasets into images. In their work, with a more recent CICIDS2017 NIDS dataset from the Canadian Institute for Cybersecurity [44], the data were transformed into a set of $50 \times 5 \times 3$ RGB images. Five attack types, Botnet, DDoS, Infiltration, Web Attack, and Portscan were considered, with Portscan designated as the source domain while each of the other four attack types were taken as the target domain data. To train the source domain model, transformed input data were fed into a four-layered CNN architecture using A batch size of 32, Stochastic Gradient Descent (SGD) optimizer, and 100 epochs. Evaluation of the model was performed on 50% validation split achieving over 95% validation accuracy and a loss of about 0.3. To transfer the knowledge from source domain to target domain, the architecture of the source domain model was altered by adding an additional dense layer with 32 units before a minimal fine-tuning of the source domain model was made. The evaluation result yields exceptional validation accuracies of 100% for both DDoS and Infiltration attack types

and about 95% for Botnet and Web attacks. The limitation of this work lies in their failure to check the in-domain performance of the target domains prior to domain adaptation.

Wu et al. [45] in their work also transformed the data into a format that could be treated like a computer vision problem but instead of having both source and target domain in the same feature space, they have introduced a heterogeneous feature scenario. Their approach was to build individual CNN NIDS models of domains and then transfer knowledge across domains by concatenating CNN source (base) model and CNN target model. They based their experiments on UNSW-NB15 of Australian Centre for Cyber Security (ACCS) [46] as the source domain dataset and NSL-KDD dataset of Canadian Institute for Cybersecurity [47] as the target. They used for their evaluation a dedicated NSL-KDD test dataset (KDDTest+ and KDDTest-21) which has 17 attacks not present in the source domain dataset. Results show that their approach recorded a validation accuracy of 87.30% on the KDDTest+ dataset versus 81.94% on the KDDTest-21 dataset. This outperformed the ordinary ConvNet without transfer learning at an accuracy of 84% on KDDTest+ dataset versus 59.92% on KDDTest-21 dataset.

In [48], a domain adaptive host-based intrusion detection method for the homogeneous feature scenario was developed. Two datasets from the Australian Defense Force Academy ADFA were used; ADFA-WD with zero-day attacks as the source domain and ADFA-WD:SAA with stealth attacks as the target domain datasets. We built an LSTM model using the source domain dataset and using varying portions of the target domain dataset; we fine-tuned the source domain model. A significant improvement in Area Under the Curve (AUC) from 83% to 91% was recorded, when we fine-tuned the source domain model with as little as 20% of the target domain dataset. The limitation of this work lies in the fact that the ADFA datasets were generated from the Windows XP operating system which as of April 2014 stopped receiving technical support from Microsoft [49]. Furthermore, the choice of AUC as a performance metric may not be the best as according to [50], "it weights omission and commission errors equally" among other problems. Based on our literature search, there are no other works on HIDS with transfer learning approach.

All the works on transfer learning-based intrusion detection reviewed in this section were able to demonstrate the benefit of transferring knowledge from one domain to the other but they all just made an assumption of similarity or dissimilarity of the source and target domains. No empirical justification was offered to show how the choice of source and target domain pairs were made. In this work, we propose a methodical approach that measures the similarity across domains before attempting transfer of knowledge between similar domains.

4. Proposed Method

The aim of our proposed method is to identify a source domain D_S that is in the best position for knowledge transfer to a given target domain D_T from among a set of domains and then leveraging knowledge available in the D_S to improve the performance of the D_T. In this section, we provide a detailed explanation of our proposed method.

4.1. Problem Setting

Before going into the problem definition, it is important to define some fundamental terms and how it has been used in this work.

Definition 1 (**Domain**). *A Domain D as used in this work denotes an attack scenario.*

Brute-force login (improper restriction of excessive authentication attempts) listed as CWE-307 in the Common Weakness Enumeration (CWE) (https://cwe.mitre.org/ (accessed on 20 September 2021)) list and Nginx Integer Overflow Vulnerability listed as CVE-2017-7529 in the Common Vulnerabilities and Exposures (CVE) (https://www.cve.org/ (accessed on 20 September 2021)) list are examples of attack scenarios.

Definition 2 (Trace). *Given a specific domain D_i, a (system call) Trace data contains the sequence of system call invocations during the execution of a program and can provide insight into whether the execution of the program was benign or malicious.*

Operating systems have a list of system calls in a way similar to how natural languages have a list of words (terms) in their vocabulary. Given an operating system with an entire list of system calls W, domain D_i will only contain W_{D_i}, a subset of W.

Definition 3 (Label). *Given a domain D and an instance of trace data, a label denotes the judgment of a domain expert as to whether the trace represents a normal execution of the program or a malicious one.*

Given a set of domains $\{D_1, D_2, D_3, \ldots, D_n\}$ each with corresponding in-domain models recording a level of performance (e.g., accuracy), it is common to find among these domains some with inadequate performance, so finding ways to improve the performance of such domains is desirable.

It can be observed that while each of the domains have their list of invoked system calls $\{W_{D_1}, W_{D_2}, W_{D_3}, \ldots, W_{D_n}\}$, there are some varying degree of overlaps between two domains D_i and D_j with corresponding list of system calls W_{D_i} and W_{D_j}. This overlap can be absolute, partial (intersection) or nothing (disjoint). For example, in the dataset LID-DS of Leipzig University, domains CWE-307 (Brute-force Login) and CWE-2014-0160 (Heartbleed) both have exactly the same set of system calls while domains CVE-2017-7529 (Nginx Integer Overflow Vulnerability) and CWE-307 as seen in Figure 2 have a partial overlap. System call numbers in the intersection are the domain independent (DI) features while those appearing in either of the domains only are the domain specific (DS) features.

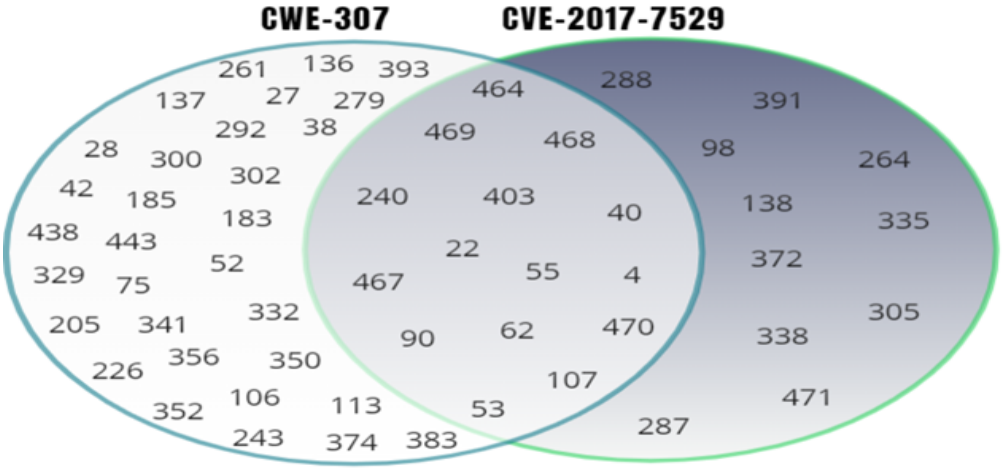


Figure 2. Illustration of overlap between domains.

To solve the problem of inadequate performance in some domains, we propose a framework which aims to achieve two subtasks:

- Source domain selection.
- Improving the performance of the target domain by leveraging the selected source domain.

In the first subtask, we are going to create in-domain models for D so as to isolate D_T and then compute domain similarity scores of all possible domain pairs to expose a candidate D_S. In the second subtask, we apply different word embedding layer combination techniques in a way to transfer the knowledge of token representation from the D_S model to the D_T model. Finally, we apply transfer learning by fine-tuning the saved in-domain D_T model.

4.2. Source Domain Selection

In this section, we use a hypothetical example to explain the source domain selection process as illustrated in Figure 3. We have assumed in this work that there is going to be at least one domain D_T whose in-domain performance is below some acceptable performance threshold. Generally, the choice of a performance threshold will be based on the situation of the in-domain models and how much improvement is desired.

Suppose we have D, a set of five elements $\{D_1, D_2, D_3, D_4, D_5\}$ which represents different attack scenarios, our goal is to identify a candidate D_S from among potential source domains. Our approach in this work is to use the knowledge in a similar, well-performing D_S to improve the performance of a domain D_T whose model is not performing well.

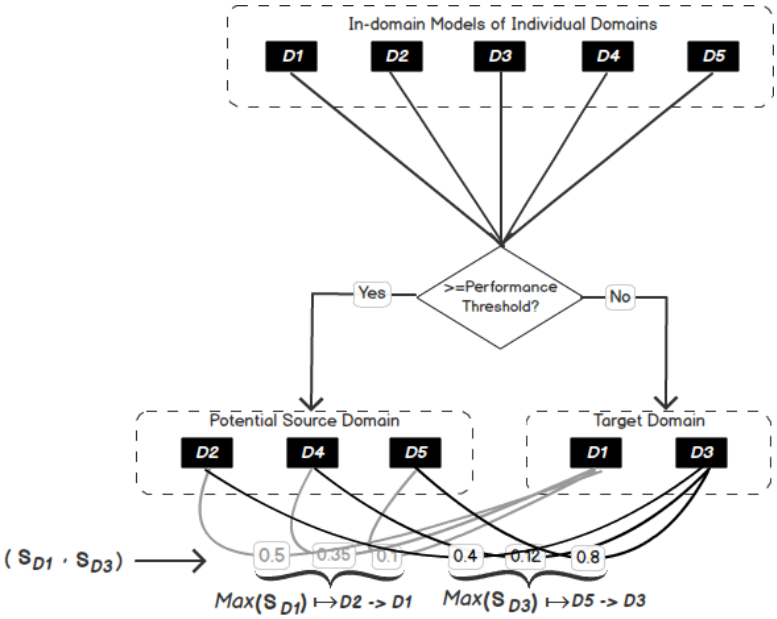


Figure 3. Source domain selection process flow.

As shown in Figure 3, this involves building in-domain models for each domain in D and based on a predefined performance threshold, grouping the domains into potential source domains D_{PS} (domains with performance equal to or greater than the threshold) and a set of target domain(s). Then for each possible source/target domain pairs $D_{PS} \rightarrow D_T$, a similarity score S is computed using Algorithm 1. The domain pair with the largest S has the candidate source domain D_S. For example, D_1 is a target domain in Figure 3 with possible domain pairs $\{D_2 \rightarrow D_1, D_4 \rightarrow D_1, D_5 \rightarrow D_1\}$, Max($S_{D_1}$) maps to domain pair $D_2 \rightarrow D_1$. This means D_2 is the selected source domain. Further details of in-domain models and domain similarity score computation are presented in Sections 4.2.1 and 4.2.2, respectively.

4.2.1. In-Domain Models

System call-based HIDS datasets, as a sequence of tokens, have been approached in other works in a Natural Language Processing (NLP) fashion [40]. NLP problems in many areas (including cybersecurity) have benefited from the application of neural network algorithms and, in recent times, deep learning has consolidated on the gains of neural networks to provide even more powerful algorithms. In Figure 4, we illustrate the general in-domain model architecture we are going to follow in this work.

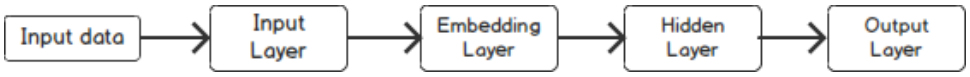


Figure 4. General In-domain model architecture.

To ensure the input layer obtains the input data in an acceptable format, some preprocessing steps were needed, specifically, tokenization, feature extraction, and padding/truncating. Tokenization involves breaking down sequences of terms into tokens which are then given numeric representation. In this work, we represent each term (system call) by their serial number in the system call table (http://manpages.ubuntu.com/manpages/focal/man2/syscalls.2.html (accessed on 20 September 2021).) For example, system call "_llseek" in the list has a representation of 1. To keep things simple, we are going to use an n-gram as our choice for feature extraction in this work, setting $n = 1$. Finally, padding/truncating is needed because sequences are usually of variable length and our case in this work is not different. We are going to set a maximum length beyond which sequences become truncated and zero pad sequences shorter than the maximum length. The situation in our dataset will guide us into picking the correct maximum length that does not lead to loss of important data.

The embedding layer implements the token embedding for the model. Token embedding presents a real-valued low-dimensional vector representation of tokens such that terms of similar meaning are made to stay close to each other in the vector space and far apart otherwise. The quality and quantity of data would have a huge impact on how well the embedding layer is trained. In this work, our embedding consists of the list of system calls of the operating system (vocabulary) $\mathcal{W}$. During training, a domain model only optimizes the embedding layer weights of the tokens present in the domain. The dimension of the embedding vector space and details about other components of Figure 4 are hyperparameters, values of which are going to be set in the course of the experiment.

In our previous paper [48], Long Short-term Memory (LSTM) was our choice of deep learning architecture. In this work, we have opted for One-dimensional Convolutional Neural Networks (1D-CNN) over other text classification deep learning algorithms, primarily due to its faster computation [51]. 1D-CNN is similar to the more popular 2D-CNN, known for its excellent performance on computer vision tasks. Unlike 2D-CNN that extracts 2D patches from image tensors and applies an identical transformation to every patch, 1D-CNN extracts local 1D patches from sequences [52] as shown in Figure 5. Setting of hyperparameter values are based on hyperparameter tuning which is a data-driven process. Manual hyperparameter tuning has been used in this work to avoid the high computing cost of other optimization methods. This will be discussed briefly in Section 5 after introducing the data we are using for our experiment.

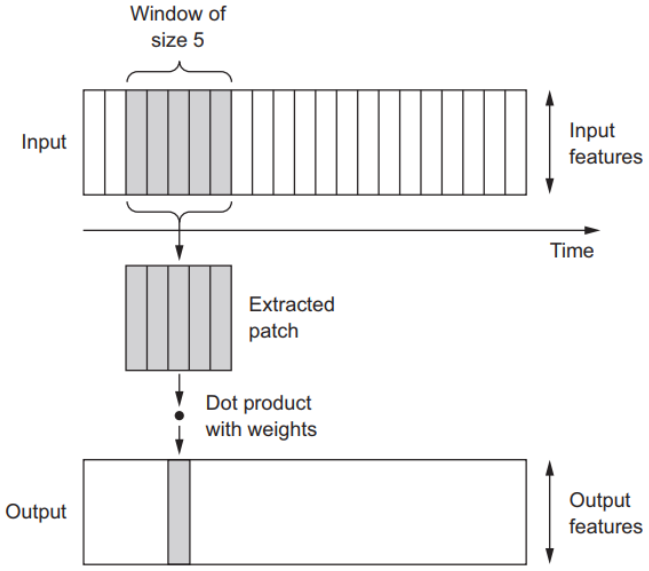


Figure 5. How 1D-CNN Works: each output timestep is obtained from a temporal patch in the input sequence [52].

4.2.2. Domain Similarity Score Computation

The purpose of the domain similarity score is to empirically measure the closeness between domains. To achieve this, we consider the frequency of occurrence of each token (system call) in a domain in relation to the frequency of occurrence in all the domains as a means of measuring the importance of tokens in each domain. As shown in Algorithm 1, the procedure is supplied with D which contains a set of domains, and for each domain, the traces are concatenated so that each domain can be taken as a document.

Algorithm 1 Domain Similarity Score Process

input: $D \leftarrow \{D_1, D_2, D_3, ..., D_n\}$
output: S **(a new** $n \times n$ **matrix)**

$E \leftarrow []$ *an empty list*
for $d \in D$ **do**
  for $t \in d$ **do**
    $E[d].extend(t)$
  end for
end for
score $\leftarrow$ *TF-IDF(E)*
$S \leftarrow$ *CosineSimilarity(score, score)*

We then use Term Frequency-Inverse Document Frequency (TF-IDF) method [53] as represented in Equation (1), to compute the score (weight) of individual tokens in each domain.

$$w_{t,d} = tf_{t,d} \times log(\frac{n}{df_t}) \tag{1}$$

$w_{t,d}$ = weight or score of term t in document d
$tf_{t,d}$ = number of occurrences of term t in document d
df_t = number of documents containing term t
n = total number of documents.

Finally, we used cosine similarity [54] to measure S, the closeness between all possible pairs of domains. Suppose we have two vectors (domain pairs) of attributes, A and B, the Cosine Similarity is represented as Equation (2).

$$CosineSimilarity = S_c(A, B) := cos(\theta) = \frac{\mathbf{A}.\mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}} \tag{2}$$

4.3. Target Domain Improvement

Now that a D_S has been selected for the D_T in needs of performance improvement, we are going to consider methods of leveraging the knowledge available in the D_S before fine-tuning the D_T. As stated earlier, token embedding is able to capture the representation of the relationship between tokens. To a large extent, the quality of the embedding layer can affect the model performance. In [55], the main idea behind their work was to take advantage of different domain-specific embeddings by combining them in order to capture the peculiarities of a given application domain as closely as possible. We take a similar approach of combining embedding layers of D_S and D_T in this work. As this may involve generating unique token list for domains, we created a function described in Algorithm 2.

We achieve target domain improvement through the following techniques:

- D_S Model Embedding Layer (Src_Emb): Here, the embedding layer of D_T model is simply replaced with the embedding layer of D_S model and frozen (Algorithm 3) so the weights of the embedding layer are not adjusted during fine-tuning. The D_T model then is fine-tuned using the train portion of D_T data.

Algorithm 2 Unique Token List

input: $D \leftarrow \{D_1, D_2, D_3, ..., D_n\}$
output: $UniqueList$

function uniqueToken(D)
  $UniqueList \leftarrow []$
  for $d \in D$ **do**
    for $t \in d$ **do**
      if $t \notin UniqueList$ **then**
        $UniqueList.extend(t)$
      end if
    end for
  end for
  return $UniqueList$
end function

Algorithm 3 D_S Model Embedding Layer (Src_Emb)

input: $Model_{D_S}, Model_{D_T}$
output: $Model_{D_{Tnew}}$

$Model_{D_{Tnew}} \leftarrow Model_{D_T}$
$Model_{D_{Tnew}}[EmbLayer] \leftarrow Model_{D_S}[EmbLayer]$
$Model_{D_{Tnew}}[EmbLayer].Freeze()$

- Domain Independent Tokens of D_S Embedding Layer (DIT_Src_Emb): In the previous technique, our assumption was that the representation of domain independent (DI) tokens in the D_S model embedding layer is so strong that the unoptimized domain specific (DS) tokens will not matter. This assumption can be true for situations where D_T has a much greater number of DI tokens than DS tokens. In situations where this is not the case, it may be better to retain the representation of the DS tokens in D_T model. This technique addresses such situation by only replacing the DI tokens representation in D_T model embedding with DI tokens representation in D_S model embedding (Algorithm 4). As with the previous technique, the derived embedding layer is frozen and D_T model is fine-tuned using the train portion of D_T data.

Algorithm 4 Domain Independent Tokens of D_S Embedding Layer (DIT_Src_Emb)

input: $Model_{D_S}, Model_{D_T}, D_S, D_T$
output: $Model_{D_{Tnew}}$

$DITokens \leftarrow uniqueToken(D_S) \cap uniqueToken(D_T)$
$Model_{D_{Tnew}}[EmbLayer] \leftarrow Model_{D_T}[EmbLayer]$
for $i \in DITokens$ **do**
  $Model_{D_{Tnew}}[EmbLayer][i] \leftarrow Model_{D_S}[EmbLayer][i]$
end for
$Model_{D_{Tnew}}[EmbLayer].Freeze()$

- Combination of Embedding Layers using PCA (Comb_PCA): One of their methods for fusing word embeddings in [55] involved concatenating embedding layers and using principal component analysis (PCA) to reduce the dimension of the derived embedding layer. We adopt the same approach here but only apply this to the DI tokens. DS tokens in D_T model are retained (Algorithm 5). Again the derived embedding layer is frozen and D_T model is fine-tuned using the train portion of D_T data.

Algorithm 5 Combination of Embedding Layers using PCA (Comb_PCA)

input: $Model_{D_S}, Model_{D_T}, D_S, D_T, EmbDim$
output: $Model_{D_{Tnew}}$

$EmbConcat \leftarrow Model_{D_S}[EmbLayer] \& Model_{D_T}[EmbLayer]$
$Pca \leftarrow PCA(EmbDim)$
$EmbPca \leftarrow Pca.Fit\_Transform(EmbConcat)$
$Model_{D_{Tnew}}[EmbLayer] \leftarrow Model_{D_T}[EmbLayer]$
$DITokens \leftarrow uniqueToken(D_S) \cap uniqueToken(D_T)$
for $i \in DITokens$ **do**
    $Model_{D_{Tnew}}[EmbLayer][i] \leftarrow EmbPca[i]$
end for
$Model_{D_{Tnew}}[EmbLayer].Freeze()$

5. Experimental Results and Analysis

5.1. Datasets

LID-DS is a modern host-based anomaly intrusion detection system (HIDS) data set recorded on a modern operating system (Ubuntu 18.04) and consists of different attack scenarios (each of the 10 scenarios represents a real vulnerability) [56]. In addition to the system call sequence, the dataset also contains metadata such as parameters, return values, user ids, process/thread ids, file system handles, timestamps, and io buffers.

In this work, we are not going to use ZipSlip as it does not have a definite CVE/CWE id attached to it. To make things compact going forward, attack scenarios will be aliased as follows: CVE-2014-0160 (CVE_2014), CWE-434 (PHP_CWE), CWE-307 (CWE_307), CWE-89 (CWE_89), CWE-434 (EPS_CWE), CVE-2012-2122 (CVE_2012), CVE-2017-7952 (CVE_2017), CVE-2018-3760 (CVE_2018) and CVE-2019-5418 (CVE_2019). We take the attack class as the positive and the benign class as the negative.

5.2. Performance Metric

The most common performance metric is Accuracy, which estimates the ratio of correctly classified instances to the entire sample size as represented in Equation (3). However, accuracy only serves as a good metric for samples that contains balanced classes. LID-DS is clearly an imbalanced data. As seen from the Benign and Attack columns of Table 2, the ratio of attacks to benign traces for all the attack scenarios is around 1:10.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3}$$

TP = True Positive
TN = True Negative
FP = False Positive
FN = False Negative.

Table 2. LID-DS Datasets.

Attack Scenario	CVE/CWE	Benign	Attack
Heartbleed	CVE-2014-0160	1000	100
PHP file upload: unrestricted upload of file with dangerous type	CWE-434	1009	103
Bruteforce login: improper restriction of excessive authentication attempts	CWE-307	994	98
SQL injection with sqlmap	CWE-89	978	100
ZipSlip	various	1000	100
EPS file upload: unrestricted upload of file with dangerous type	CWE-434	972	99
MySQL authentification bypass	CVE-2012-2122	1240	155
Nginx integer overflow vulnerability	CVE-2017-7952	983	174
Sprockets information leak vulnerability	CVE-2018-3760	1084	137
Rails file content disclosure vulnerability	CVE-2019-5418	981	98

Precision and Recall are two popular performance metrics used in imbalanced dataset situations [57]. As shown in Equation (4), Precision measures the proportion of samples with positive labels that are indeed positive.

$$Precision = \frac{TP}{TP + FP} \tag{4}$$

While in Equation (5), Recall measures the proportion of the positive class samples labeled correctly by the model.

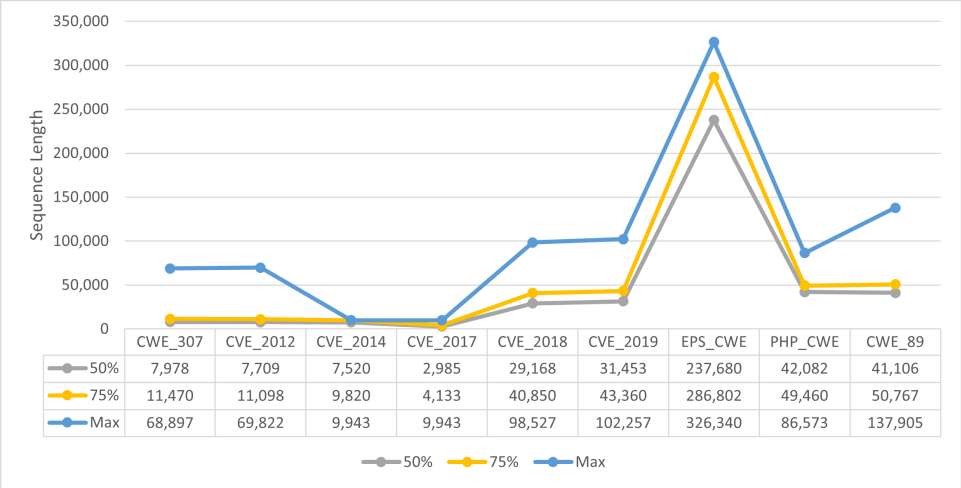
$$Recall = \frac{TP}{TP + FN} \tag{5}$$

To combine the effects of Precision and Recall into one performance metric, F-*Measure* in Equation (6) computes the harmonic mean of these two important performance metrics. In this work, we are going to set $\beta = 2$ because we are more interested in recall than precision.

$$F_\beta = (1 + \beta^2) \cdot \frac{Precision \cdot Recall}{(\beta^2 \cdot Precision) + Recall} \tag{6}$$

5.3. In-Domain Models

As stated in Section 4.2.1, our deep learning architecture of choice is 1D-CNN. It is important that we have our input as a sequence of fixed length but we must ensure we are not losing too much information as we try to achieve this. To have a better understanding of our data, we represent the variation of sequence length across domains as a quartile and in Figure 6 we present line plots of median, third quartile and fourth quartile.



	CWE_307	CVE_2012	CVE_2014	CVE_2017	CVE_2018	CVE_2019	EPS_CWE	PHP_CWE	CWE_89
50%	7,978	7,709	7,520	2,985	29,168	31,453	237,680	42,082	41,106
75%	11,470	11,098	9,820	4,133	40,850	43,360	286,802	49,460	50,767
Max	68,897	69,822	9,943	9,943	98,527	102,257	326,340	86,573	137,905

Figure 6. Domain Sequence length Plot: 50% Line Represents the Median, the 75% line represents the third quartile, and max represents the fourth qaurtile.

We noticed that the sequence length varies from one domain to the other. It can be observed that five domains (CWE_307, CVE_2012, CVE_2018, CVE_2019 and CWE_89) have their fourth qaurtile sequence length greater than their third quartile sequence length by more than double while the other domains have their fourth quartile sequence length close to the third. For all domains, the median and the third quartile sequence lengths are very close to each other. Consequently, we set a maximum sequence length equal to the third quartile length for each domain. The preliminary experiment shows that this is slightly better than median length in terms of model performance.

Using manual hyperparameter tuning, we settled for the following hyperparameter values:

- Embedding dimension: 32
- Sequence Maximum Length: third quartile of the domain sequence length
- number of 1D-CNN layers: 2
- number of dense layers: 2
- MaxPooling1D: 5
- Kernel Size: 3
- batch size: 50
- Number of Epochs: 50

Based on these hyperparameter values, we built a 1D-CNN model architecture. The visualization of the 1D-CNN network architecture is presented in Figure 7.

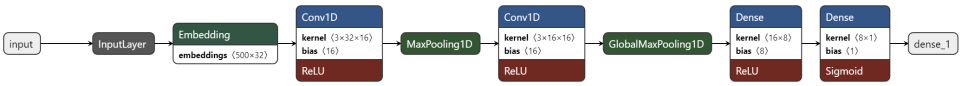


Figure 7. 1D-CNN model architecture visualization.

For each domain, in-domain models were built using the training portion. In a stratified manner, we used a train-test split ratio of 75:25. We also built an extra model that we named LID using a training dataset which is created by concatenating the training portion of all the individual domains. To address the data imbalance problem, we adjusted class weights for each of the domains. We achieved this in TensorFlow Keras by passing class_weight parameter, which was calculated for a binary classification scenario based on the formula in Equation (7). Each of the in-domain model was saved so they can be reused.

$$w_0, w_1 = \frac{n_0 + n_1}{2n_0}, \frac{n_0 + n_1}{2n_1} \tag{7}$$

w_0 = weight of class 0
w_0 = weight of class 1
n_0 = number of instances in class 0
n_0 = number of instances in class 1.

As shown in Table 3, models of almost all the domains recorded F2-score greater than 90%, except for CWE_307 and CVE_2014. It is, therefore, reasonable to set a performance threshold of 90% as required for the source domain selection process (Figure 3). This leaves us with CVE_2014 with F2-score of 13% and CWE_307 with F2-score of 80% as our target domains. All other domains are potential source domains D_{PS}. Then we compute similarity scores ($S_{CWE\_307}, S_{CVE\_2014}$) for each possible potential source/target domain pairs $D_{PS} \rightarrow D_T$.

Table 3. In-doman Models Result.

Test Set	Precision	Recall	F2-Score
CWE_307	0.48	0.96	0.8
CVE_2012	1.00	1.00	1.00
CVE_2014	0.21	0.12	0.13
CVE_2017	1.00	1.00	1.00
CVE_2018	1.00	1.00	1.00
CVE_2019	1.00	1.00	1.00
PHP_CWE	1.00	0.96	0.97
CWE_89	1.00	1.00	1.00
EPS_CWE	1.00	1.00	1.00
LID	0.96	0.91	0.92

5.4. Similarity Score

By implementing Algorithm 1, we are able to compute similarity scores for all possible domain pairs. The algorithm assumes that every domain could be the source and every domain could be the target so the result as seen in Table 4 presents the similarity scores of every possible pairs. Looking at the fourth row of Table 4, it can be observed that CVE_2014 records a similarity score of 0.935 when paired with CWE_307 which suggests a very close similarity. To make sense of this, we inspected data of these two domains and discovered that they both contain exactly the same set of system calls. We also noticed in Figure 6 that they both have about 50% of their samples with sequence length less than 8000.

Table 4. Domain similarity scores of all possible domain pairs.

Scenarios	CWE_307	CVE_2012	CVE_2014	CVE_2017	CVE_2018	CVE_2019	EPS_CWE	PHP_CWE	CWE_89
CWE_307	1	0.431	0.935	0.328	0.056	0.058	0.564	0.379	0.373
CVE_2012		1	0.311	0.050	0.093	0.100	0.346	0.366	0.369
CVE_2014			1	0.323	0.052	0.054	0.479	0.439	0.426
CVE_2017				1	0.038	0.037	0.187	0.169	0.173
CVE_2018					1	1.000	0.052	0.042	0.041
CVE_2019						1	0.054	0.046	0.045
EPS_CWE							1	0.138	0.133
PHP_CWE								1	0.998
CWE_89									1

Other works observed the similarity between some of the attack scenarios. In a recent work using Siamese-CNN for building a multi-class model on LID-DS [58], it was observed through the confusion matrix that (CWE_307, CVE_2014), (CVE_2018, CVE_2019) and (PHP_CWE,CWE_89) were not properly classified because of the similarity between them. According to [58], (PHP_CWE,CWE_89), for example, could be grouped together as they were identified by the Open Web Application Security Project (OWASP) as vulnerabilities in which a hacker can transmit hostile data to the interpreter. It is of note that all these domain pairs have a similarity score of approximately 1.

5.5. Target Domain Improvement

As seen in Figure 3, similarity scores S of pairing D_T with each of the D_{PS} is required in order to select the candidate D_S for a particular D_T. The D_S that produced the highest similarity score $Max(S_T)$ when paired with D_T is taken as the candidate D_S. Given that we have D_T: D_{CWE_307} and D_{CVE_2014}, we can deduce from Table 4 that $Max(S_{CWE\_307})$ with a score of 0.564431 points to candidate D_S: D_{EPS_CWE} and $Max(S_{CVE\_2014})$ with a score of 0.479089 also points to candidate D_S: D_{EPS_CWE}.

In our experiment, for each D_T, we picked D_{LID} and the domains with the top three similarity scores from among D_{PS} as D_S. Therefore, for target domain D_{CWE_307}, we are evaluating the following source domains: D_{LID}, D_{EPS_CWE}, D_{CVE_2012} and D_{PHP_CWE}. Furthermore, for the target domain D_{CVE_2014}, we are evaluating D_{LID}, D_{EPS_CWE}, D_{PHP_CWE} and D_{CWE_89} as our source domains. We apply the techniques outlined in Section 4.3 to each of the source/target domain pair.

Figures 8 and 9 present the results of our experiments. It can be observed from the results of both target domains that the embedding layer of the source domain D_{LID} consistently had better impact on improving the target domains. This makes sense as the embedding layer produced while training domain D_{LID} has been optimized through access to larger and more diverse training data.

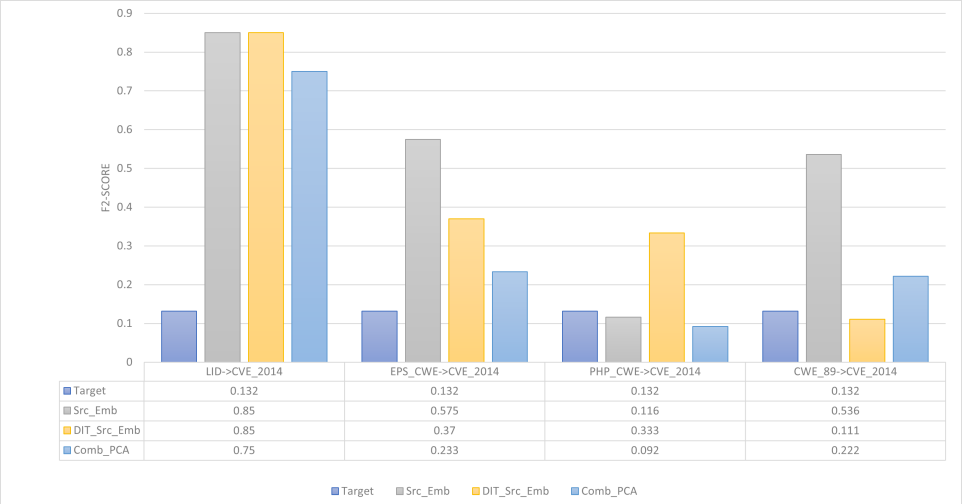


Figure 8. Experimental results of pairing different source domains with target domain heartbleed (CVE-2014-0160) attack scenario.

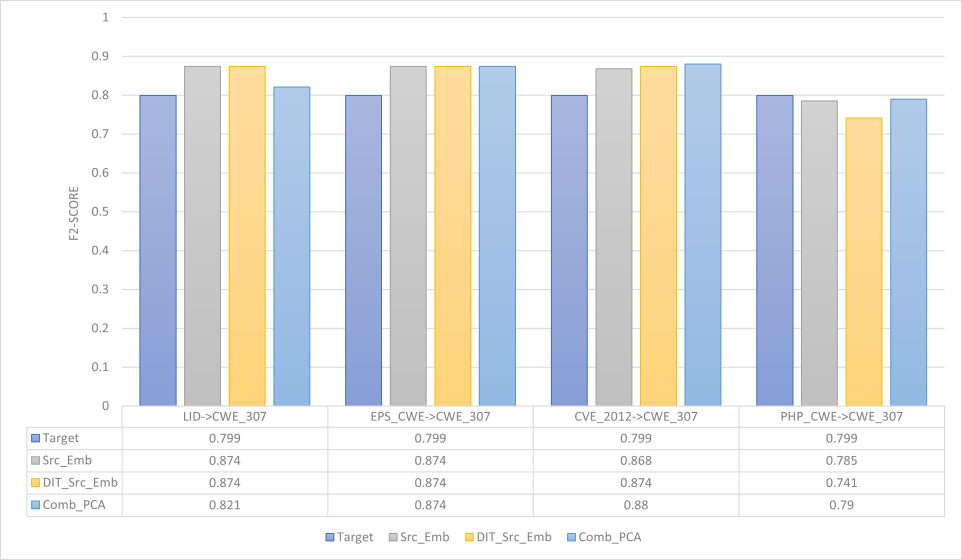


Figure 9. Experimental results of pairing different source domains with target domain bruteforce login (CWE-307) attack scenario.

Furthermore, the other three source domains for each of the target domains produced improvement of the target domain that is consistent with their similarity scores, especially in the application of DIT_Src_Emb technique. For example, when paired with D_{CVE_2014}, potential source domain D_{EPS_CWE} recorded the highest similarity score of 0.479089, D_{PHP_CWE} recorded the second-highest score of 0.439176 and D_{CWE_89} recorded the third-highest score of 0.425669. As seen in Figure 8, the performance improvement of D_{CVE_2014} resulting from these three source domains with DIT_Src_Emb technique are (EPS_CWE->CVE_2014:0.37, PHP_CWE->CVE_2014:0.33 and CWE_89->CVE_2014:0.11).

It appears that the Comb_PCA technique is only promising if the target domain model performance is well above average. As seen in Figure 9, the performance of Comb_PCA is on par with other techniques. Finally, results show that application of Src_Emb technique is the most rewarding in terms of target domain performance improvement except for PHP_CWE->CVE_2014 where Src_Emb recorded an F2-score of 0.116 which is even less than the target domain F2-score.

6. Conclusions and Future Directions

In this paper, developing cross-domain host-based intrusion detection was proposed. We developed a method for source domain model selection by quantifying the amount of similarity existing between individual attack domains. Then, using different word embedding space modification techniques and transfer learning, we leverage the knowledge from a well performing attack source domain to improve the performance of a similar attack target domain. Our experiment on LID-DS using the in-domain approach exposed two straggling domains, CWE_307 with 80% F2-score and CVE_2014 with 0.13. Using our source domain selection method, we were able to select the top three source domain models from which we can transfer some knowledge to improve the straggling target domains. For the selected source/target domain pair, we applied embedding space modification techniques and observed improvement of the target domain which is consistent with the similarity score of the source domains.

In the future, we will look into developing cross-domain network intrusion detection. Ultimately, we will carry out engineering implementation of the cross-domain intrusion detection ideas from this paper and the network version we are working on, for attack detection in IoT/edge security especially collaborative uncrewed assets and smart devices. Lastly, in order to take advantage of similarities between attack scenarios of different operating systems, we will explore the cross-lingual intrusion detection direction.

Author Contributions: Conceptualization, O.A., A.G. and R.F.E.; Methodology, O.A. and A.G.; Software, O.A.; Validation, A.G., R.F.E. and C.B.; Formal Analysis, O.A.; Investigation, O.A.; Resources, R.F.E. and C.B.; Data Curation, O.A.; Writing—Original Draft Preparation, O.A.; Writing—Review & Editing, A.G.; Visualization, O.A.; Supervision, A.G., R.F.E. and C.B.; Funding Acquisition, R.F.E. and C.B. All authors have read and agreed to the published version of the manuscript.

Funding: This research is supported by the United States Army Research Laboratory (ARL) under the Grant No. W911NF2120076.

Data Availability Statement: Publicly available datasets were analyzed in this study. This data can be found here [https://github.com/LID-DS/LID-DS#lid-ds-leipzig-intrusion-detection---data-set (accessed on 20 September 2021)].

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Snabe Hagemann, J.; Weinelt, B. Digital Transformation of Industries: Demystifying Digital and Securing $100 Trillion for Society and Industry by 2025. Available online: https://issuu.com/laszlopacso/docs/wef-digital-transformation-of-indus (accessed on 26 September 2022).
2. Zhu, J.; McClave, E.; Pham, Q.; Polineni, S.; Reinhart, S.; Sheatsley, R.; Toth, A. *A Vision Toward an Internet of Battlefield Things (IoBT): Autonomous Classifying Sensor Network*; Technical Report; US Army Research Laboratory: Adelphi, MD, USA, 2018.
3. Narayanan, P.; Vindiola, M.; Park, S.; Logie, A.; Waytowich, N.; Mittrick, M.; Richardson, J.; Asher, D.; Kott, A. *First-Year Report of ARL Directors Strategic Initiative (FY20-23): Artificial Intelligence (AI) for Command and Control (C2) of Multi-Domain Operations (MDO)*; Technical Report; US Army Combat Capabilities Development Command, Army Research Laboratory: Adelphi, MD, USA, 2021.
4. Parkavi, R.; Nithya, R.; Priyadharshini, G. Digital Terrorism Attack: Types, Effects, and Prevention. In *Critical Concepts, Standards, and Techniques in Cyber Forensics*; IGI Global: Hershey, PA, USA, 2020; pp. 61–87.
5. Morgan, S. *2019 Official Annual Cybercrime Report*. Available online: https://cybernetsecurity.com/industry-papers/CV-HG-2019-Official-Annual-Cybercrime-Report.pdf (accessed on 26 September 2022).
6. Marelli, M. The SolarWinds hack: Lessons for international humanitarian organizations. *Int. Rev. Red Cross* **2022**, *104*, 1–18. [CrossRef]
7. Crane, C. Cyber Attack Statistics By Year: A Look at the Last Decade, 42. Available online: https://sectigostore.com/blog/42-cyber-attack-statistics-by-year-a-look-at-the-last-decade/ (accessed on 26 September 2022).
8. Ponemon, I. *Cost of Data Breach Report 2021*. Available online: https://www.dataendure.com/wp-content/uploads/2021_Cost_of_a_Data_Breach_-2.pdf (accessed on 26 September 2022).
9. Morgan, S. Cybercrime to Cost the World $10.5 Trillion Annually by 2025. Available online: https://cybersecurityventures.com/cybercrime-damage-costs-10-trillion-by-2025/ (accessed on 26 September 2022).

10. Hindy, H.; Atkinson, R.; Tachtatzis, C.; Colin, J.N.; Bayne, E.; Bellekens, X. Utilising deep learning techniques for effective zero-day attack detection. *Electronics* **2020**, *9*, 1684. [CrossRef]
11. Modi, C.; Patel, D.; Borisaniya, B.; Patel, H.; Patel, A.; Rajarajan, M. A survey of intrusion detection techniques in cloud. *J. Netw. Comput. Appl.* **2013**, *36*, 42–57. [CrossRef]
12. Mishra, P.; Varadharajan, V.; Tupakula, U.; Pilli, E.S. A detailed investigation and analysis of using machine learning techniques for intrusion detection. *IEEE Commun. Surv. Tutorials* **2018**, *21*, 686–728. [CrossRef]
13. Tidjon, L.N.; Frappier, M.; Mammar, A. Intrusion detection systems: A cross-domain overview. *IEEE Commun. Surv. Tutorials* **2019**, *21*, 3639–3681. [CrossRef]
14. Khraisat, A.; Gondal, I.; Vamplew, P.; Kamruzzaman, J. Survey of intrusion detection systems: techniques, datasets and challenges. *Cybersecurity* **2019**, *2*, 1–22. [CrossRef]
15. De Boer, P.; Pels, M. Host-Based Intrusion Detection Systems. Available online: https://www.os3.nl/_media/2004-2005/rp1/report19.pdf (accessed on 26 September 2022).
16. Vigna, G.; Kruegel, C. Host-Based Intrusion Detection System. Available online: https://susy.mdpi.com/user/manuscripts/resubmit/c279999d25191388cc075c6d0cb5ce2a (accessed on 26 September 2022).
17. Li, Y.; Xia, J.; Zhang, S.; Yan, J.; Ai, X.; Dai, K. An efficient intrusion detection system based on support vector machines and gradually feature removal method. *Expert Syst. Appl.* **2012**, *39*, 424–430. [CrossRef]
18. Sarker, I.H. Machine learning: Algorithms, real-world applications and research directions. *Comput. Sci.* **2021**, *2*, 1–21. [CrossRef]
19. Kim, K.; Aminanto, M.E. Deep learning in intrusion detection perspective: Overview and further challenges. In Proceedings of the 2017 International Workshop on Big Data and Information Security (IWBIS), Jakarta, Indonesia, 23–24 September 2017; pp. 5–10.
20. Aminanto, E.; Kim, K. Deep learning in intrusion detection system: An overview. In Proceedings of the 2016 International Research Conference on Engineering and Technology (2016 IRCET), Higher Education Forum, Bali, Indonesia, 28–30 June 2016.
21. Gangopadhyay, A.; Odebode, I.; Yesha, Y. A Domain Adaptation Technique for Deep Learning in Cybersecurity. In Proceedings of the OTM Confederated International Conferences "On the Move to Meaningful Internet Systems", Rhodes, Greece, 21–25 October 2019.
22. Bleiweiss, A. LSTM Neural Networks for Transfer Learning in Online Moderation of Abuse Context. In Proceedings of the 11th International Conference on Agents and Artificial Intelligence, Prague, Czech Republic, 19–21 Feburary 2019.
23. Mou, L.; Meng, Z.; Yan, R.; Li, G.; Xu, Y.; Zhang, L.; Jin, Z. How transferable are neural networks in nlp applications? *arXiv* **2016**, arXiv:1603.06111.
24. Braud, C.; Lacroix, O.; Søgaard, A. Cross-lingual and cross-domain discourse segmentation of entire documents. *arXiv* **2017**, arXiv:1704.04100.
25. Pan, S.J.; Ni, X.; Sun, J.T.; Yang, Q.; Chen, Z. Cross-domain sentiment classification via spectral feature alignment. In Proceedings of the 19th International Conference on World Wide Web, Raleigh, NC, USA, 26–30 April 2010.
26. Buczak, A.L.; Guven, E. A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Commun. Surv. Tutorials* **2015**, *18*, 1153–1176. [CrossRef]
27. Kreibich, C.; Crowcroft, J. Honeycomb: creating intrusion detection signatures using honeypots. *Acm Sigcomm Comput. Commun. Rev.* **2004**, *34*, 51–56. [CrossRef]
28. Cannady, J. Artificial Neural Networks for Misuse Detection. Available online: http://pld.cs.luc.edu/courses/intrusion/fall05/cannady.artificial_neural_networks_for_misuse_detection.pdf (accessed on 26 September 2022).
29. Livadas, C.; Walsh, R.; Lapsley, D.; Strayer, W.T. Usilng machine learning technliques to identify botnet traffic. In Proceedings of the 2006 31st IEEE Conference on Local Computer Networks, Tampa, FL, USA, 14–16 November 2006; pp. 967–974.
30. Kruegel, C.; Toth, T. Using decision trees to improve signature-based intrusion detection. In Proceedings of the International Workshop on Recent Advances in Intrusion Detection, Pittsburgh, PA, USA, 8–10 September 2003; pp. 173–191.
31. Norton, M.; Roelker, D. Snort 2.0 Rule Optimizer. *Sourcefire Netw. Secur. White Pap.* Available online: https://www.cs.ucdavis.edu/~wu/ecs236/sf_snort20_detection_rvstd.pdf (accessed on 26 September 2022).
32. Gharibian, F.; Ghorbani, A.A. Comparative study of supervised machine learning techniques for intrusion detection. In Proceedings of the Fifth Annual Conference on Communication Networks and Services Research (CNSR'07), Fredericton, NB, Canada, 14–17 May 2007; pp. 350–358.
33. The Third International Knowledge Discovery and Data Mining Tools Competition Dataset KDD Cup 1999 Data. 1999. Available online: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html (accessed on 20 September 2021).
34. Kruegel, C.; Mutz, D.; Robertson, W.; Valeur, F. Bayesian event classification for intrusion detection. In Proceedings of the 19th Annual Computer Security Applications Conference, Las Vegas, NV, USA, 8–12 December 2003; pp. 14–23.
35. Wagner, C.; François, J.; Engel, T.; State, R. Machine learning approach for ip-flow record anomaly detection. In Proceedings of the International Conference on Research in Networking, Valencia, Spain, 9–13 May,2011.
36. Brauckhoff, D.; Wagner, A.; May, M. FLAME: A Flow-Level Anomaly Modeling Engine. Available online: https://www.usenix.org/legacy/event/cset08/tech/full_papers/brauckhoff/brauckhoff_html/ (accessed on 26 September 2022).
37. Zhang, J.; Zulkernine, M.; Haque, A. Random-forests-based network intrusion detection systems. *IEEE Trans. Syst. Man Cybern. Part (Appl. Rev.)* **2008**, *38*, 649–659. [CrossRef]

38. Wang, Y.; Yang, K.; Jing, X.; Jin, H.L. Problems of kdd cup 99 dataset existed and data preprocessing. *Appl. Mech. Mater.* **2014**, *667*, 218–225. [CrossRef]
39. Simon, C.K.; Sochenkov, I.V. Evaluating Host-Based Intrusion Detection on the adfa-wd and ADFA-WD: SAA Datasets. 2021. Available online: Semanticscholar.org (accessed on 20 September 2021).
40. Vinayakumar, R.; Alazab, M.; Soman, K.; Poornachandran, P.; Al-Nemrat, A.; Venkatraman, S. Deep learning approach for intelligent intrusion detection system. *IEEE Access* **2019**, *7*, 41525–41550. [CrossRef]
41. Park, D.; Ryu, K.; Shin, D.; Shin, D.; Park, J.; Kim, J. A Comparative Study of Machine Learning Algorithms Using LID-DS DataSet. *Kips Trans. Softw. Data Eng.* **2021**, *10*, 91–98.
42. Xu, Y.; Liu, Z.; Li, Y.; Zheng, Y.; Hou, H.; Gao, M.; Song, Y.; Xin, Y. Intrusion Detection Based on Fusing Deep Neural Networks and Transfer Learning. In Proceedings of the International Forum on Digital TV and Wireless Multimedia Communications, Shanghai, China, 19–20 September 2019.
43. Divekar, A.; Parekh, M.; Savla, V.; Mishra, R.; Shirole, M. Benchmarking datasets for anomaly-based network intrusion detection: KDD CUP 99 alternatives. In Proceedings of the 2018 IEEE 3rd International Conference on Computing, Communication and Security (ICCCS), Kathmandu, Nepal, 25–27 October 2018; pp. 1–8.
44. Sharafaldin, I.; Lashkari, A.H.; Ghorbani, A.A. Toward generating a new intrusion detection dataset and intrusion traffic characterization. *ICISSp* **2018**, *1*, 108–116.
45. Wu, P.; Guo, H.; Buckland, R. A transfer learning approach for network intrusion detection. In Proceedings of the 2019 IEEE 4th International Conference on Big Data Analytics (ICBDA), Suzhou, China, 15–18 March 2019; pp. 281–285.
46. Moustafa, N.; Slay, J. UNSW-NB15: A comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). In Proceedings of the 2015 Military Communications and Information Systems Conference (MilCIS), Canberra, Australia, 10–12 November 2015; pp. 1–6.
47. Tavallaee, M.; Bagheri, E.; Lu, W.; Ghorbani, A.A. A detailed analysis of the KDD CUP 99 data set. In Proceedings of the 2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications, Ottawa, ON, Canada, 8–10 July 2009; pp. 1–6.
48. Ajayi, O.; Gangopadhyay, A. DAHID: Domain Adaptive Host-based Intrusion Detection. In Proceedings of the 2021 IEEE International Conference on Cyber Security and Resilience (CSR), Rhodes, Greece, 26–28 July 2021; pp. 467–472.
49. Shilov, A. Microsoft's Windows XP Finally Dead: Last Embedded Version Reaches EOL. Available online: https://www.anandtech.com/show/14200/microsofts-windows-xp-finally-dead-last-embedded-version-reaches-eol#:~:text=Microsoft's%20Windows%20XP%20Home%20and,EOL%20on%20January%208%2C%202019 (accessed on 20 September 2021).
50. Lobo, J.M.; Jiménez-Valverde, A.; Real, R. AUC: a misleading measure of the performance of predictive distribution models. *Glob. Ecol. Biogeogr.* **2008**, *17*, 145–151. [CrossRef]
51. Kiranyaz, S.; Avci, O.; Abdeljaber, O.; Ince, T.; Gabbouj, M.; Inman, D.J. 1D convolutional neural networks and applications: A survey. *Mech. Syst. Signal Process.* **2021**, *151*, 107398. [CrossRef]
52. Chollet, F. *Deep Learning with Python*; Simon and Schuster: New York, NY, USA, 2021.
53. Christian, H.; Agus, M.P.; Suhartono, D. Single document automatic text summarization using term frequency-inverse document frequency (TF-IDF). *ComTech Comput. Math. Eng. Appl.* **2016**, *7*, 285–294. [CrossRef]
54. Xia, P.; Zhang, L.; Li, F. Learning similarity with cosine similarity ensemble. *Inf. Sci.* **2015**, *307*, 39–52. [CrossRef]
55. Rettig, L.; Audiffren, J.; Cudré-Mauroux, P. Fusing vector space models for domain-specific applications. In Proceedings of the 2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI), Portland, OR, USA, IEEE, 4–6 November 2019; pp. 1110–1117.
56. Grimmer, M.; Röhling, M.M.; Kreusel, D.; Ganz, S. A modern and sophisticated host based intrusion detection data set. *IT-Sicherh. Voraussetzung Eine Erfolgreiche Digit.* 2019; pp. 135–145. Available online: https://www.researchgate.net/profile/Martin-Grimmer/publication/357056160_A_Modern_and_Sophisticated_Host_Based_Intrusion_Detection_Data_Set/links/61b9faaf1d88475981f04cb9/A-Modern-and-Sophisticated-Host-Based-Intrusion-Detection-Data-Set.pdf (accessed on 26 September 2022).
57. Johnson, J.M.; Khoshgoftaar, T.M. Survey on deep learning with class imbalance. *J. Big Data* **2019**, *6*, 27. [CrossRef]
58. Park, D.; Kim, S.; Kwon, H.; Shin, D.; Shin, D. Host-Based Intrusion Detection Model Using Siamese Network. *IEEE Access* **2021**, *9*, 76614–76623. [CrossRef]