CADE: Detecting and Explaining Concept Drift Samples for Security Applications (Supplementary Materials)

1 Feature Engineering Details

We describe the detailed steps to perform feature engineering on the selected datasets.

Drebin. We followed the original paper [1] and used one-hot encoding to construct the feature vectors. As we mentioned, to be realistic, we only used the features (*e.g.*, strings, permissions, APIs) that appeared in the training data. The original feature dimensionality is between 5,055 and 7,296 (depending on which class is chosen as the unseen family). Since the feature vectors are very sparse, we reduced the dimensionality to about 1,000 based on the variance of each feature. We used the VarianceThreshold function in the scikit-learn package [2] to perform the dimension reduction.

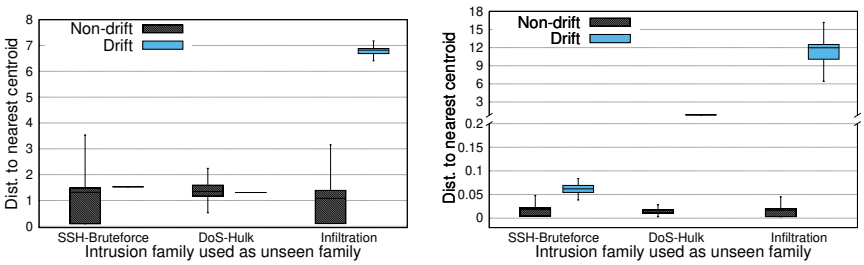
IDS2018. Each sample is a network flow, and the features are computed based on both directions: forward flow and backward flow [3]. Each sample has 80 features originally, two of which are categorical features (*i.e.* "Dst Port" and 'Protocol'), and the rest are numerical features. We used one-hot encoding for the categorical features. "Dst Port" is mapped into three categories based on its frequency of appearance (high, medium, and low). We did not encode the port number directly because the vector will be too high-dimensional. Specifically, if the destination port appears more than 10,000 times, it would be mapped to a high-frequency port. Destination port shows between 1,000 and 10,000 times are regarded as medium-frequency port. The rest of the ports are mapped to low-frequency port. "Protocol" is also mapped to a three-dimensional vector based on its value (TCP, UDP, IPv6). For the numerical features, we used MinMaxScaler in the scikit-learn package [2] to normalize them within $[0,1]$.

After the pre-processing, each sample is a vector of 83 dimensions, where each feature is within $[0,1]$. Similar to the Drebin dataset, we also built the encoding dictionary and the MinMaxScaler based on the training set and applied them to the testing samples. We mapped the unseen values of the categorical features to all zeros, indicting these values are not exist in the training set. Note that the feature "Timestamp" is originally represented by a string, we transform the string into milliseconds, which are numerical values. We will release the pre-processed datasets along the final version of the paper.

2 Distance-based Explanation using Gradient Method

Our distance-based explanation method in `CADE` is perturbation-based. An alternative way of designing the distance-based explanation method is to use *gradients*. Intuitively, after approximating the detection boundary, we can compute the partial derivative of the approximation model output with respect to the input and identify the features with the highest gradients. The higher gradients indicate that perturbing these features will have a stronger influence upon the latent distance between the drifting sample and the nearest centroid. However, this gradient-based method involves approximating the boundary as well as the direction to maximize the distance changes. To reduce the errors by the approximation, we can directly compute the direction that triggers the largest distance change. Since the latent space learned by the contrastive learning is a Euclidean space, this direction refers to $f(\boldsymbol{x}_t) - \boldsymbol{c}_{y_t}$, the direction from a drifting sample $\boldsymbol{x}_t$ to its nearest centroid $\boldsymbol{c}_{y_t}$. As such, we identify the important features by computing the gradient of $f(\boldsymbol{x}_t) - \boldsymbol{c}_{y_t}$ with respect to $\boldsymbol{x}_t$ and selecting the features with the highest gradients.

We run an evaluation of this idea and choose the same number of important features as `CADE`. Using the evaluation metrics introduced in the main paper (*i.e.* fidelity and the ratio of perturbed samples cross the boundary), we find this idea is not working well. More specifically, the gradient-based method shows a similar performance with the *random baseline*, indicating it fails to identify important features. We suspect the reason is that gradient can only reflect feature sensitivity to a small perturbation. However, we need a relatively large perturbation to impose a significant influence on the distance. As such, perturbation-based explanations are more suitable



(a) Original space

(b) Latent space (CADE)

Figure 1: Boxplot of the distances between testing samples and their nearest centroids in both the *original space* and *latent space* for the IDS2018 dataset. Samples from previously unseen family are regarded as drifting samples.

for our problem than gradient-based explanations.

3 IDS2018 Additional Results

Figure 1 shows the distance of the (non-)drifting samples to their closest centroid in the input space and the latent space of CADE. The results are aligned with those on the Drebin dataset, confirming that contrastive learning helps to separate drifting samples from non-drifting ones.

In Table 1 we present a case study on the explanation results of a drifting sample in IDS2018. We use the setting when *Infiltration* is the unseen family and randomly pick one drifting sample for explanation. According to CADE, the closest family is *DoS-Hulk*. After running the explanation, we locate 18 features.

We manually examine the selected features in Table 1, and look up the documentation of IDS2018 [3] to determine if the features have relevant semantic meanings. Specifically, *Infiltration* means the attacker first sends a malicious file via an email to the victim to exploit the victim's host. If successful, attackers will further run portscan (*e.g.*, using Nmap) and exploit more vulnerabilities [3]. *DoS-Hulk* is a particular type of Denial of Service (DoS) attack that aims to over-load the targeted host with superfluous network requests.

Different from Drebin (whose features only have binary values), features in the IDS2018 dataset could be numerical. To interpret the meaning of the features, we also compare the sample's feature values with those of the centroid of *DoS-Hulk*. We use $\uparrow$ (or $\downarrow$) to indicates whether the Infiltration flow has a higher (or lower) feature value.

We find most selected features make intuitive sense. These features are mainly describing the port-scan activities caused by the Infiltration. Some features are about the characteristics of the DoS attacks. First, the FSH flag is frequently set in the Infiltration flow ("FSH Flag Cnt") which indicates port scanning (*e.g.*, XMAS scan). The next four features show Infiltration backward flow's inter-arrival time ("BWD IAT") is shorter. This makes sense because DoS attack will cause large delays to backward flow as the target host is overwhelmed. It also makes sense that Infiltration flow's "Down/Up Ratio" is higher since DoS attack barely has any down-flow traffic. Then the lower "Fwd Pkts/s" indicates that the Infiltration forward flow is sending packets not as fast as DoS. Then the features on the next row are mostly showing the packet sizes of Infiltration are larger than those of DoS-Hulk. Interestingly, Infiltration has sent more packets in total (due to the portscan). Overall, the selected features are useful to explain why the infiltration sample is different from DoS-Hulk attacks. The only mis-selected feature is the "Timestamp" feature, which only indicates the two attacks happened at a different time (not about attack characteristics).

Appendix-G: The Impact of Concept Drift on Supervised Classifier

We use the two datasets to examine how drifting samples impact the accuracy of the original classifier. We trained an MLP multi-class classifier using the training set. Then we test the trained MLP classifier on the testing sets with and without the previously unseen family. As shown in Table 2, when the testing set does not have any previously-unseen family, the original classifier performs well with an average accuracy of over 99.7% for both datasets. Then if we add the previously unseen families into the testing set (we iteratively test each family as the unseen family), the overall accuracy drops below 62%. As such, it is necessary to monitor incoming samples to detect drifting samples.

Appendix-H: Drifting Detection with Prediction Probability

We provide additional evidence to show that the softmax outputted prediction probability is not a good indicator for drifting samples. We construct a baseline for drifting detection by ranking this prediction probability. We take the testing samples with a lower softmax prediction probability as drifting samples. Similar to Transcend, we ignore testing samples

IDS2018 Case-B: Drifting Sample Family: **Infiltration**; Closest Family: **DoS-Hulk**
PSH Flag Cnt ↑ , Bwd IAT Tot ↓ , Bwd IAT Mean ↓ , Bwd IAT Min ↓ , Flow IAT Max ↓ , Down/Up Ratio ↑ , Fwd Pkts/s ↓ , Bwd Seg Size Avg ↑ , Bwd Pkt Len Max ↑ , Pkt Len Mean ↑ , Pkt Len Std ↑ , Pkt Size Avg ↑ , Fwd Seg Size Avg ↑ , Fwd Pkt Len Max ↑ , Subflow Fwd Byts ↑ , Tot Fwd Pkts↑ , Flow Byts/s ↑ , Timestamp ↑.

Table 1: Case study of explaining why a given sample a drifting sample. The highlighted features represent those that match the semantic characteristics that differentiate the drifting sample with the closest family. ↑ means the drifting sample has a higher feature value compared to the closest family (DoS-Hulk); ↓ means the drifting sample has a lower feature value. "Pkt": packets; "Seg": segment; "Fwd": forward; "Bwd": backward; "Tot": total; "IAT": inter-arrival time.

Testing Set	**Drebin** Avg ± Std	**IDS2018** Avg ± Std
Testing set (w/o drifting)	0.9976 ± 0.00	1.0000 ± 0.00
Testing set (w/ drifting)	0.6201 ± 0.18	0.5602 ± 0.20

Table 2: Average accuracy of the original classifier on the testing sets (with and without the drifting samples).

Metrics	**Drebin (Avg±Std)**	**IDS2018 (Avg±Std)**
Precision	0.95 ± 0.08	0.50 ± 0.39
Recall	0.88 ± 0.11	0.67 ± 0.47
F1	0.91 ± 0.06	0.56 ± 0.41
Norm. Effort	0.94 ± 0.17	1.70 ± 0.49

Table 3: The average drifting detection results for Drebin and IDS2018 datasets using the softmax outputted probability. For each evaluation metric, we report the mean value and the standard deviation.

with a prediction probability of 1.0 (*i.e.*, they are not considered as drifting). We report the average results on Drebin and IDS2018 datasets in Table 3. We can observe that the softmax outputted probability works OK on the Drebin dataset (91% of F1 score) but only achieves 56% of F1 score on the IDS2018 dataset, validating its inefficiency on detecting drifting samples.

References

[1] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, Konrad Rieck, and CERT Siemens. Drebin: Effective and explainable detection of android malware in your pocket. In *Proc. of NDSS*, 2014.

[2] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 2011.

[3] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A Ghorbani. Toward generating a new intrusion detection dataset and intrusion traffic characterization. In *Prof. of ICISSP*, 2018.