Efficient Malware Analysis Using Metric Embeddings

ETHAN M. RUDD, DAVID KRISILOFF, and SCOTT COULL, Mandiant Inc., USA
DANIEL OLSZEWSKI, University of Florida, USA
EDWARD RAFF, Booz Allen Hamilton, USA
JAMES HOLT, Laboratory for Physical Sciences, University of Maryland, USA

Real-world malware analysis consists of a complex pipeline of classifiers and data analysis—from detection to classification of capabilities to retrieval of unique training samples from user systems. In this article, we aim to reduce the complexity of these pipelines through the use of low-dimensional metric embeddings of Windows PE files, which can be used in a variety of downstream applications, including malware detection, family classification, and malware attribute tagging. Specifically, we enrich labeling of malicious and benign PE files with computationally-expensive, disassembly-based malicious capabilities information. Using this enhanced labeling, we derive several different types of efficient metric embeddings utilizing an embedding neural network trained via contrastive loss, Spearman rank correlation, and combinations thereof. Our evaluation examines performance on a variety of transfer tasks performed on the EMBER and SOREL datasets, demonstrating that low-dimensional, computationally-efficient metric embeddings maintain performance with little decay. This offers the potential to quickly retrain for a variety of transfer tasks at significantly reduced overhead and complexity. We conclude with an examination of practical considerations for the use of our proposed embedding approach, such as robustness to adversarial evasion and introduction of task-specific auxiliary objectives to improve performance on mission critical tasks.

CCS Concepts: • **Security and privacy** → **Malware and its mitigation**; • **Computing methodologies** → **Transfer learning**; **Supervised learning;**

Additional Key Words and Phrases: Metric embeddings, malware analysis, transfer learning, multi-objective learning, deep learning

ACM Reference format:
Ethan M. Rudd, David Krisiloff, Scott Coull, Daniel Olszewski, Edward Raff, and James Holt. 2024. Efficient Malware Analysis Using Metric Embeddings. *Digit. Threat. Res. Pract.* 5, 1, Article 4 (March 2024), 20 pages.
https://doi.org/10.1145/3615669

4

1 INTRODUCTION

Malware analysis is a complex process involving highly-skilled experts and many person-hours of effort. Given the number of new files seen each day (more than 500,000 on VirusTotal alone [44]), automation of malware analysis is a necessity in any real-world setting. The development of new analysis tools provides an avenue for more efficient malware analysis teams.

Fortunately, malware analysis tasks are often amenable to **machine learning** (**ML**) solutions. The tasks (e.g., malware detection or malware family classification) are complex enough that traditional rules-based approaches

Authors' addresses: E. M. Rudd, D. Krisiloff, and S. Coull, Mandiant Inc. 11951 Freedom Drive, 6th Floor, Reston, Virginia, 20190, United States of America; e-mails: ethan.rudd@mandiant.com, david.krisiloff@mandiant.com, scott.coull@mandiant.com; D. Olszewski, University of Florida, 601 Gale Lemerand Dr, Gainesville, FL 32611, United States of America; e-mail: dolszewski@ufl.edu; E. Raff, Booz Allen Hamilton, 10025 Governor Warfield Pkwy, Columbia, MD 21044, United States of America; e-mail: raff_edward@bah.com; J. Holt, Laboratory for Physical Sciences, University of Maryland, 8050 Greenmead Dr, College Park, MD 20740, United States of America; e-mail: holt@lps.umd.edu.

2576-5337/2024/03-ART4 $15.00
https://doi.org/10.1145/3615669

remain brittle and require frequent updating. At the same time, it is possible to acquire and label large datasets to train ML models using threat feeds and crowdsourcing services, like VirusTotal or Reversing Labs. This combination of large-scale datasets and powerful ML models leads to complex malware pipelines that can automate most stages of the analysis process, including detection, capabilities classification, and retrieval of unique samples for future training.

One notable downside, however, is that ML models come with significant technical debt: they need to be retrained as malware evolves and often interdependencies between various components of the analysis pipeline can be hard to understand or predict [38]. Even with a consistent, reusable feature vector representation for these pipeline components, training on industry-scale datasets may require tens of terabytes of storage and model re-training can take multiple weeks.

This motivates the question: what if we can use ML to derive low-dimensional representations which capture semantic behavior of malware/goodware that can be used to reduce complexity and resource requirements for downstream tasks? This could significantly enhance capability for training classifiers for novel applications, performing rapid iteration/experimentation, and efficiently updating deployed models, all while reduced processing and storage requirements.

Since other applications of applied ML, including biometrics and **information retrieval** (**IR**) systems, have utilized metric learning to derive low-dimensional embeddings for similar downstream tasks, in this article, we explore whether we can apply metric learning in a similar vein toward various malware analysis-oriented ML tasks. Using metric embeddings, we aim to simplify some of the engineering costs associated with running and maintaining a state-of-the-art malware analysis pipeline.

Contrary to other applications of ML classification, where data can trivially be assigned labels corresponding to one or more classes/attributes, labeling for malware analysis tasks can be more difficult [50]. Moreover, data for malware analysis often includes telemetry and metadata beyond hard labels, which could ideally be used to enrich our metric embeddings. In this article, we explore techniques to enrich our embeddings with complex semantic information provided by computationally-expensive tools (e.g., disassembly). This allows us to explore whether it is possible to approximate more expensive analysis with lower-overhead static representations.

When generating our embeddings, we utilize Mandiant's CAPA tool; an open source tool, which utilizes rules and heuristics in conjunction with disassembly to yield capability labels (e.g., file read/write, registry key generation, process creation, data send/receive over networks, socket connection, base64/XOR encoding) associated with a given PE, ELF, and .NET file as well as shellcode snippets. We enrich samples from the EMBER dataset with these computationally-expensive capability labels, and using these labels generate different types of efficient embeddings, including a Siamese embedding, which utilizes a contrastive loss over clusters of CAPA attributes, as well as a novel ranking embedding, which uses the Spearman rank correlation coefficient as a loss and aims to embed ranked degree of similarity between different CAPA attribute clusters. We then perform comparisons of these different metric embedding loss functions across two different datasets: EMBER and SOREL-20M, on three different downstream transfer tasks: malware detection, malware family classification and malware attribute classification, making comparisons to original dataset benchmarks where applicable. Finally, we perform an analysis of practical considerations surrounding the use of our metric embeddings, including robustness to adversarial evasion, qualitative analysis of the underlying learned metric space, and the use of task-specific auxiliary loss functions to improve performance on critical tasks.

The contributions of our article are summarized as follows:

(1) We are the first to propose a metric learning approach that incorporates semantically-rich info into an efficiently-computable metric space for malware analysis tasks.
(2) We show a number of novel training regimes for the model using a Siamese network, including contrastive, Spearman, and task-specific losses. In doing so, we are the first to combine contrastive and Spearman losses to provide both coarse and fine-grained similarity information during training.

(3) We demonstrate that generic metric embeddings can successfully tackle several important malware analysis tasks, including detection, family classification, and type prediction.
(4) We evaluate the adversarial robustness of metric embeddings in the malware analysis problem space, which has not been previously explored.

2 BACKGROUND AND RELATED WORK

Malware analysis involves a plethora of tasks which security vendors aim to automate. These tasks may include detection, capabilities determination, authorship attribution, malware family categorization, and similarity search, among many others. Malware analysis tasks can be approached in a variety of ways, including via ML or via some form of signature, heuristic, or rule-based approach. Moreover, these tasks can be approached using raw bytes, a disassembled representation, or via dynamic emulation. In general, static detection based on featurization of raw bytes is highly efficient and can be applied in less restrictive contexts with less overhead, while disassembly and dynamic emulation may offer higher precision for specific samples, but are less efficient and less conducive to automation including large-scale ML (trained on hundreds of millions of samples). In practice, malware analysis tools utilize a number of different approaches, which leads to significant technical debt. The approach that we discuss in this manuscript aims to derive compact and efficient representations based on static features, which are enriched from additional telemetry and can be applied in a fast and efficient manner to a multitude of downstream malware analysis use cases, thus reducing technical debt.

2.1 Metric Learning

Metric learning is a ML task that focuses on learning distances (i.e., metrics/measures) between objects that captures some semantically-meaningful notion of similarity. These learned metric functions play an important role in fields including IR, ranking, and recommendation systems [23]. The key property of a similarity metric is that it maps similar objects close together and dissimilar objects far apart within the learned metric space. In practice, the objects are represented by a set of features, the metric function is the transformation of the features into a common metric space, and the learning process finds a transformation such that the similarity/dissimilarity behavior is correct with respect to the labels provided during training. Various learning architectures have been proposed, including Siamese networks that learn from the distances between pairs of objects, and triplet networks that use three samples to capture both similarity and dissimilarity to an anchor [24], [19]. There are also different loss functions for each architecture along with other subtle modifications of the learning process that can be applied to improve results (e.g., specialized algorithms for stochastic gradient descent [23]).

2.2 Binary File Similarity

To perform metric learning, we require a dataset of objects along with their similarities. Since our objects are binary **portable executables** (**PEs**) we need to define a meaningful notion of similarity among binaries. Binary similarity gauges the likeness between two binary files and can be defined in several different ways. Perhaps the cleanest definition is that two binaries are similar if they were compiled from the same source code or contain a large fraction of the same source code [40], [46]. This definition is particularly useful from a malware reverse engineering standpoint: knowing that a file contains source code from a known piece of malware can significantly speed up analysis. However, this definition introduces problems when labeling a dataset of binary files built from unknown source code, which is the case in practical malware analysis settings.

Our enrichment approach uses Mandiant's CAPA tool [3] to provide capability telemetry for assessing similarity between two malware/goodware samples in a metric learning loss function. Other methods which in some form summarize code capabilities and attributes could be employed in lieu of CAPA, including other code similarity measures based on disassembly [14, 15, 29] or function call-graphs derived from dynamic analysis [9, 26, 48] would be viable alternatives to CAPA.

Another alternative, which we utilize in Section 5.1, is to use hash-based similarity measures. Hash-based similarity measures have been used for assessing similarity between samples, and performing a number of downstream malware analysis tasks, including finding near-duplicate samples, clustering, and family attribution. Oliver et al. [30] introduced the T-LSH algorithm which uses a sliding window over byte strings to tabulate an array of counts. Winter et al. introduce a strategy based on n-grams contained in piecewise hash signatures that allows for efficient code similarity search [45]. Breitinger et al. introduced other methods utilizing majority voting of piecewise hashes in conjunction with Bloom filters and run-length encoding to obtain efficient and compressed code summaries for similarity search and retrieval [5, 7, 8]. In Reference [6], the authors extend the approach to identify similar parts of a file. Additional efficiency improvements have been presented in other work, including [27]. Raff and Nicholas introduce an approach using the Lempel-Ziv Jaccard Distance in Reference [33] and extend this approach to malware classification in Reference [32]. Hash-based similarity measures are fast and low-overhead, and can be applied to several downstream malware analysis tasks. However, they intrinsically involve no ML step, which limits their generalization to certain downstream tasks, particularly when changes in the population of malware occurs.

2.3 Metric Learning for Malware Analysis

Prior work has explored metric learning for malware analysis, and they generally fall into two types. In the first type (Type 1), a standard pre-established metric learning approach (e.g., contrastive, triplet) is applied to features from static malware binary samples with pre-existing clean labels. The second type of approach (Type 2) uses disassembly or dynamically derived features to provide richer context. Standard metric learning approaches are then used to train the model.

2.3.1 Type 1 Approaches. Yang et al. [49] apply a similar evaluation methodology to ours on Android malware, training a base representation, then fitting shallow neural networks to accomplish various tasks. While their approach is interesting, the contrastive approach is adapted from a relatively stock formulation based on learning from positive pairs of samples with dropout applied. There is no further enrichment capability within the loss. Moreover, the datasets that the authors evaluate on are several times smaller than ours, and they operate strictly on raw text and numeric features of Android configuration files (e.g., the Android manifest), leading to a comparatively more constrained problem space.

In Reference [20], Jurecek et al. apply off-the-shelf shallow metric learning approaches to PE malware family classification. Unfortunately, without adaptation, some of these approaches are not trivially scalable to commercial data volumes. The authors experimented on a dataset of only 14K samples—two and three orders of magnitude smaller, respectively, than the EMBER and SOREL datasets that we use in this manuscript. We also note that the experiments in Reference [20] apply no form of enrichment within the metric learning loss beyond family labels. In some respects, this work builds on other studies by Jurecek and Lórencz [21, 22], which employ a weighted KNN approach, with weightings derived via particle swarm optimization. Their loss functions, unlike ours, are built on presumed binary or categorical labels, which provide no further enrichment. We would note that while [21] and [22] use a slightly larger dataset than [20], it is still proprietary and a fraction of the size of the EMBER dataset.

2.3.2 Type 2 Approaches. Liu et al. apply a graph contrastive approach to PE classification [28], with features for the graph structure derived from running each malware sample in a Cuckoo sandbox and trained using a standard contrastive loss. Chen et al. [10] introduce another approach which uses graph metric learning on disassembled PE files, this time with a triplet loss. A KNN classifier is used for malware family classification on the metric space. The authors evaluate this approach on a relatively small proprietary dataset with fewer than 20K samples in train and test. In Reference [13], Dib et al. employ contrastive learning with a strong focus on novelty detection and concept drift. This is done atop a fine-tuned BERT model on normalized instructions

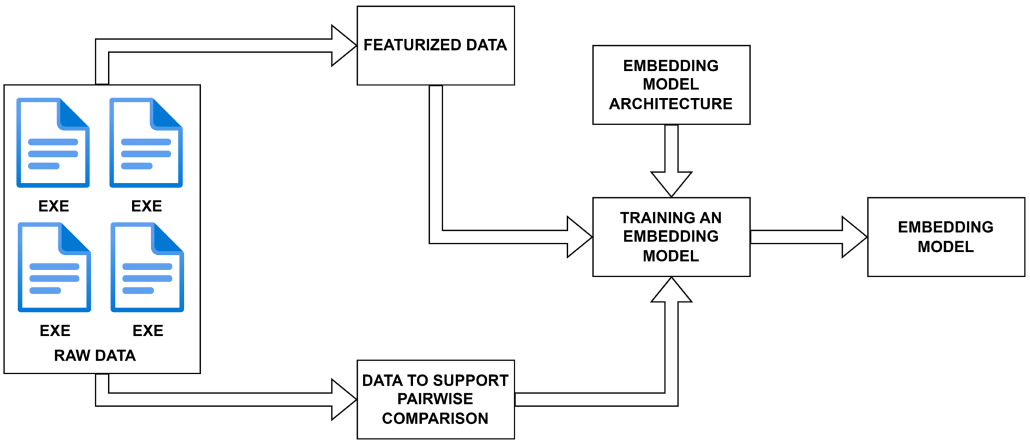


Fig. 1. A system diagram depicting the upstream training of an embedding model. Training is conducted using a selected embedding model architecture, feature vectors from executable binaries, and additional data to support pairwise sample comparisons (e.g., CAPA labels).

derived from disassembled PE files. Experiments are conducted using a custom dataset of 90K samples, and are therefore not readily reproducible.

All three approaches rely on input representations from sandbox runs or PE file disassembly for both training and inference. These approaches are better suited for cloud or post-hoc analysis workflows than endpoint AV where their compute time is prohibitively large.

2.3.3 Comparisons. To the best of our knowledge, ours is the first work that aims to utilize metric learning losses to enrich compact, statically derived embedding representations with capability and behavioral data. In contrast to Type 1 and Type 2 approaches, we rely on lightweight, endpoint viable features, while learning a metric from the rich semantic knowledge of disassembly. Both Type 1 and Type 2 approaches fail to do this. Type 1 and Type 2 approaches employ metric learning losses on presumed labels (binary, categorical, etc.), while ours utilizes derived disassembly telemetry and metric learning loss functions to enrich low-dimensional representations with information about the capabilities of the corresponding executable binaries. To some extent, Type 2 approaches learn these capabilities via a language model or graph structure. However, they require disassembly or sandbox emulation during inference, whereas our approach only requires disassembly during training, and is quick and efficient at inference time.

3 APPROACH

In this article, we focus on building a model that produces embeddings of Windows PE files. The goal is to learn a representation that can be used for multiple downstream malware analysis tasks. The methodology can be separated into two phases. Figure 1 represents the first phase where an embedding model is trained. To begin, we take the dataset of raw Windows PE binaries and apply two processes: (1) a featurization step, and (2) a step to compute file information that aids in determining the pairwise similarity between any two files. For featurization, we focus on **subject matter expert** (**SME**)-derived, static features that are efficient to compute and which capture a broad range of malicious signals. These features include things such as the APIs imported, parsing errors, entropy, and byte distributions.

To determine pairwise similarity between two PE files, we use a tool like CAPA [3] that detects capabilities of executable files (i.e., two files with overlapping capabilities are similar). Notably, we hypothesize that using more complex similarity information than what is available naturally in the features (e.g., disassembly-based

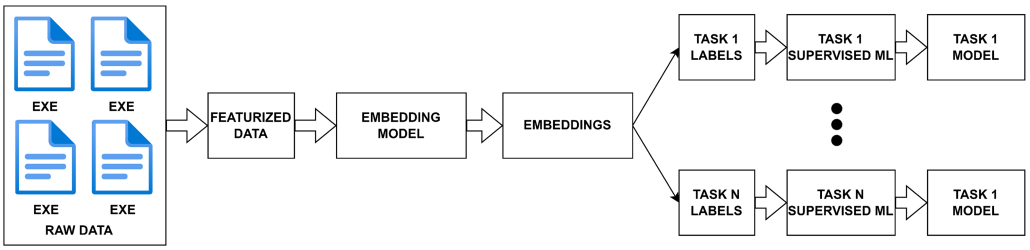


Fig. 2. A system diagram depicting the downstream use of a trained embedding model for multiple tasks. Features are extracted from executable binaries and fed as inputs to a trained embedding model, which generates a low-dimensional embedding representation of each of the input binaries. These embeddings along with auxiliary label information can be used for different downstream malware analysis tasks.

similarity vs. static features) will imbue the learned metric space with additional information without the added overhead of the more complex analysis techniques. Once the data and labels are defined, we specify an architecture for the embedding, which we instantiate with a multi-layered neural network. An algorithm then takes those components as input and trains an embedding model. The training algorithm can use a metric embedding network (e.g., a Siamese network) to learn the model parameters that effectively cause similar PE files to be near one another in the embedding space, and dissimilar files to be farther apart.

During the second phase, as shown in Figure 2, we measure the transferability of the embedding space to various malware analysis tasks. Concretely, we embed our training data and use that representation to train new models for each downstream task. By keeping the models used for the transfer process constant and only varying our embedding process, we can make precise measurements of the utility of various similarity information, loss function, binary representation, or other parameters of our embedding network. The following sections detail our modeling setup.

3.1 Network Architecture

Our embedding neural network architecture is shown in Figure 3. The network takes as input 2381-dimensional static features defined by the EMBER 2018 dataset [2], though the approach is flexible to any static features used. The features are first normalized via standard scaling with respect to mean and variance on the EMBER 2018 training set, then fed to an embedding network, which is consist of four dense layers with sigmoid activations of dimension 4,000, 1,024, 512, and 512. Between the layers, we include both a BatchNorm and a Dropout layer with a dropout probability of 10%. Network architecture and hyperparameters were derived using a hold-out dataset of malware/goodware binaries. Following those layers is the final embedding layer using a linear activation with specified output dimension; in the majority of our experiments, we utilize an output dimension of 32 unless otherwise noted. During training, the network outputs are then optionally normalized and losses, which compare CAPA attribute information, are evaluated and minimized via backpropagation.

3.2 Enriching Metric Embeddings with CAPA Labels

The CAPA system detects various capabilities of a binary file using both the static analysis and disassembly and yields a set of capabilities for each file. These capabilities are categorical and non-mutually exclusive and are labeled with short text snippets (e.g., “encode data using Base64”). We incorporate these generated sets of capabilities to enrich our embeddings via two different loss functions, which we apply both solo and in tandem (via summation) in our experiments. An example CAPA report is shown in Figure 4.

3.2.1 *Contrastive Loss.*

The Siamese contrastive loss is defined as

$$L_{contrastive} = Y_{true} D + (1 - Y_{true}) \max(margin - D, 0), \tag{1}$$

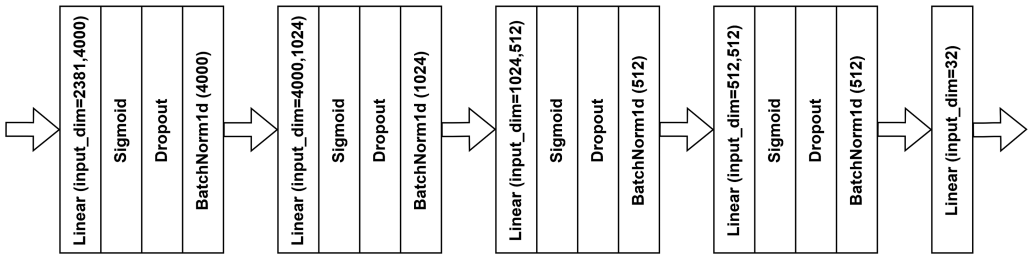


Fig. 3. Architectural schematic of our embedding network.

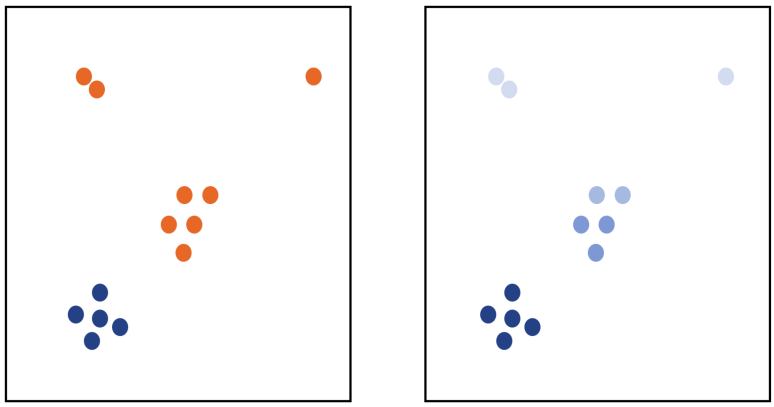
```
$ capa Lab01-01.dll_
+-----------------------+-------------------------------------------------------------------------------------------+
| md5                   | 290934c61de9176ad682ffdd65f0a669                                                          |
| path                  | Lab01-01.dll_                                                                             |
+-----------------------+-------------------------------------------------------------------------------------------+

+-----------------------------------------------------+-----------------------------------------------------------+
| CAPABILITY                                          | NAMESPACE                                                 |
|-----------------------------------------------------+-----------------------------------------------------------|
| receive data                                        | communication                                             |
| send data                                           | communication                                             |
| initialize Winsock library                          | communication/socket                                      |
| receive data on socket                              | communication/socket/receive                              |
| send data on socket                                 | communication/socket/send                                 |
| connect TCP socket                                  | communication/socket/tcp                                  |
| create TCP socket                                   | communication/socket/tcp                                  |
| act as TCP client                                   | communication/tcp/client                                  |
| check mutex                                         | host-interaction/mutex                                    |
| create mutex                                        | host-interaction/mutex                                    |
| resolve DNS                                         | host-interaction/network/dns/resolve                      |
| create process                                      | host-interaction/process/create                           |
+-----------------------------------------------------+-----------------------------------------------------------+
```

Fig. 4. A Sample CAPA report for a PE file Lab01-01.dll_ from Reference [3].

where D is the distance between a pair of points, Y_{true} is 1 if the pair contains similar objects or 0 if the objects are dissimilar, and margin is the desired separation between dissimilar objects and is a tunable hyperparameter. Equation 1 requires that samples either belong to the same group or not, meaning that we cannot include more fine-grained similarity (e.g., these two samples are 75% similar), in the loss function. Consequently, in this case, we convert the CAPA detection sets into hard clusters with a locality sensitive hash. Employing a MinHash with one band and 64 permutations, we compute a single hash (cluster) for each binary file, where two files are similar if they lie in the same cluster ($Y_{true} = 1$) and different otherwise ($Y_{true} = 0$). During our experiments, we employ a contrastive loss using Euclidean distance with a margin of 10, which was selected with light hyperparameter optimization on the training dataset.

3.2.2 Spearman Loss. While our contrastive approach assesses similarity based on CAPA clusters, it coarsely embeds binaries as "similar" or "not similar", when in reality, some sets of CAPA labels are more similar than others. To account for finer-grained similarities, we employ a novel approach based on the Spearman rank correlation coefficient.

Specifically, advances in approximate differentiable sorting and ranking [4] allow us to optimize Spearman's rank correlation coefficient with **stochastic gradient descent** (**SQD**). This allows us to compute the loss between a ground truth ranking and a predicted ranking from our model. This is desirable as it allows inserting *more nuanced information into the loss function based on finer grained degree*, rather than a simple binary similar/dissimilar decision. In our experiments, the ranking is based on similarity, from most similar to least. Given



(a) Contrastive (b) Spearman

Fig. 5. A visual representation of the difference between the contrastive and Spearman losses. The contrastive loss considers all entities, either in-cluster (blue) or out-of-cluster (orange). The Spearman loss captures the relative similarity of objects, shown in shades of blue, regardless of cluster status.

integer ranks, we can define Spearman's rank correlation coefficient as

$$r = 1 - \frac{6 \sum_i (R(X_i) - R(Y_i))^2}{n(n^2 - 1)}, \tag{2}$$

where X_i,Y_i are the ground truth similarity and predicted similarity of data point i and $R(X_i)$,$R(Y_i)$ are the corresponding ranks. We assess ground truth similarity between two CAPA capability sets as their Jaccard similarity, and use the soft rank implementation from Blondel et al. [4] to compare predicted and ground truth ranks. For a given batch, we use these ranks to establish the Spearman rank correlation coefficient—the loss for that batch. A visual depiction of the difference between contrastive and Spearman losses is shown in Figure 5.

3.3 Training Process

All the layers of our networks are initialized by the Xavier algorithm [17] and trained with SGD to a maximum of 30 epochs with a learning rate of 0.001. The batching algorithm used for SGD training was modified to better support our metric learning loss functions. Ordinarily, each batch contains C randomly sampled (with replacement) clusters and M randomly sampled PE files from each cluster (again, sampled with replacement). For our binary similarity problem, we are confronted with two complications. First, each cluster can potentially contain both goodware and malware unlike typical metric learning problems where clusters are homogeneous. Second, we have an extremely large number of clusters ($O(10^5)$). To address the goodware and malware heterogeneity concern, we split each cluster C into two clusters, C-goodware and C-malware. When we sample C clusters, we do so from the combination of all C-goodware and C-malware clusters. To address the second concern, we sample C clusters without replacement and define the end of the epoch when the model has processed examples from every cluster. This algorithm ensures we cover the full space of goodware, malware, and clusters in each epoch while maintaining a balance between positive and negative pairs in each batch. For these experiments, we set $C = 20$ and $M = 4$.

3.4 Transfer Process

After training the embedding, we measure the embedding's usability on various malware classification tasks. For our experiments, we trained an embedding network using the EMBER 2018 training partition and extracted

CAPA labels. Once we have a trained embedding network, we can use this to extract embeddings from any dataset with EMBER features. Using extracted embeddings for a given dataset, we can then fit a lightweight classifier over the embeddings and corresponding labels to make predictions for arbitrary different tasks.

The choice of the best final classifier for each task is not obvious. Typically, generalization-based learning using ensemble methods (e.g., random forests or gradient boosted trees) provides state-of-the-art performance on malware tasks. However, our feature space is unique in that distances between two training points have meaning and decision tree methods that rely on splitting individual features may have difficulty capturing that geometry. Notably, SVMs are a generalization-based method that could take our metric space into account, but we ignore it here due to the computational cost of training an SVM on very large datasets. An alternative would be an instance-based learning algorithm (e.g., k-nearest neighbors), which explicitly considers distances between training data points. As we will show in the following evaluation, we consider both instance and generalization-based classifiers, and the best classifier can vary based on the transfer task.

4 EXPERIMENTS

For our primary evaluation, we trained various embedding networks using EMBER feature vectors and CAPA labeling extracted from PEs in the EMBER 2018 [2] train partition. These consist of: (1) contrastive loss on CAPA clusters, (2) Spearman loss on Jaccard similarities between CAPA attribute sets, and mixed objective Spearman and Contrastive loss, where the net loss term is the sum of the losses. We also test a weighting of 10x on the Spearman loss term to bring the contributions from each constituent loss term to roughly the same order of magnitude.

Each embedding network is trained according to the procedure discussed in Section 3.3. Since deep learning models are not amenable to convex optimization (i.e., no global minimum guarantee), we trained five different instantiations of each model in order to assess variance in performance. When performing transfer task experiments, we then aggregated mean and standard deviation statistics across embeddings from all five networks of a given type.

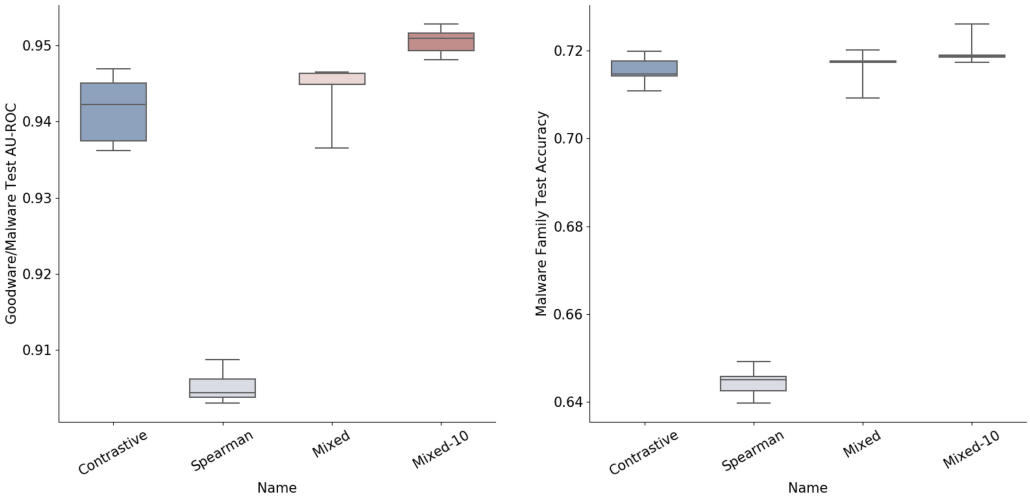
4.1 Transfer Experiments on EMBER

As an initial evaluation of our embeddings, we performed two transfer tasks on the EMBER 2018 dataset: malware detection and malware family classification.

The 2018 version of the EMBER dataset consists of PE malware and goodware features and labels. Samples are time-distributed between 2017 and 2018, with the training partition temporally preceding the test partition. The training partition consists of 300k malware samples, 300k benign samples, and 200k unlabeled samples that—consistent with the majority of papers that use EMBER—we do not utilize during training. The testing partition consists of 100k malware samples and 100k benign samples. While the precise criterion for malicious vs. benign labels is proprietary, EMBER malware family labels, which we use in our transfer experiments were derived via AVClass [39]. In order to obtain CAPA telemetry, we pulled the original PE files from a vendor aggregator and ran them through Mandiant's open source CAPA v1 tool.

The malware detection transfer task aims to detect malware using EMBER's malicious/benign labels. For this task, we extracted embeddings across both train and test partitions of EMBER 2018. We then fit a lightGBM ensemble with 1,000 trees and otherwise default parameters over the embeddings extracted from the training set, and evaluated using embeddings extracted from the test set. The results of this experiment are shown in Figure 6(a) in terms of the **area under the ROC** (**AU-ROC**) curve on the test set.

In this experimental setting, we tried different weightings of the mixed objective loss, with the Spearman component both unweighted and up-weighted by a factor of 10 to be on the same scale as the contrastive component. We notice that the transfer performance on the contrastive loss embedding significantly outperforms the transfer performance on the Spearman loss embedding. However, both embeddings which use a combination of the



(a) Goodware/Malware (b) Malware Family

Fig. 6. Transfer Experiments on EMBER. In (a), results of the Goodware/Malware transfer experiment are reported in terms of the area under the ROC curve (AU-ROC). In (b), results of the malware family transfer experiment are reported in terms of accuracy. For both experiments, classifiers trained on Spearman embeddings underperformed those trained on Contrastive embeddings while classifiers trained on embeddings derived from our weighted mixed-objective loss were the top performers.

two losses offer better classification performance than strictly either of the embeddings trained on a solo loss (Spearman or Contrastive), with the weighted mixed objective loss outperforming the unweighted. Note that none of the transfer malicious/benign classifiers on EMBER 2018 exceed the baseline model from [2].

The second transfer task is a malware family recognition task, which utilizes a 1-nearest neighbor classifier in the embedding space in conjunction with the EMBER 2018 malware family labels (derived via AVClass [39]) to predict the family for malicious samples. These results are shown in Figure 6(b). While here the performance evaluation is in terms of accuracy not AU-ROC, we notice the same performance trend across embedding types as for the detection transfer task—that is, the mixed objective equals or out performs contrastive or Spearman loss in isolation.

4.2 Transfer Experiments on SOREL-20M

We additionally evaluated the performance of our embeddings for different tasks on the SOREL-20M dataset. SOREL is a large industry-scale dataset with publicly available labeling telemetry beyond just malicious/benign detection; it also contains public labeling telemetry for 11 distinct malware attributes, namely: Adware, Crypto Miner, Downloader, Dropper, File Infector, Flooder, Installer, Packed, Ransomware, Spyware, and Worm. Note that these attributes are non-mutually exclusive across samples, meaning that a given malware sample can have multiple malware attributes. We can think of the SOREL's malware attributes as defining high-level behaviors encapsulate fine-grained capabilities, similar to those identified by CAPA.

SOREL also contains different data with a different data distribution than EMBER (on which the embeddings were extracted). This suggests any strong performance over the EMBER-to-SOREL transition is an indication of the robustness of our approach to producing general purpose representations for downstream tasks. We extracted 32-dimensional embeddings for all of the SOREL samples apriori, and trained task-specific lightGBM classifiers on the extracted embeddings. We assessed embedding performance/quality for two distinct tasks on SOREL: malware detection and malware attribute labeling.

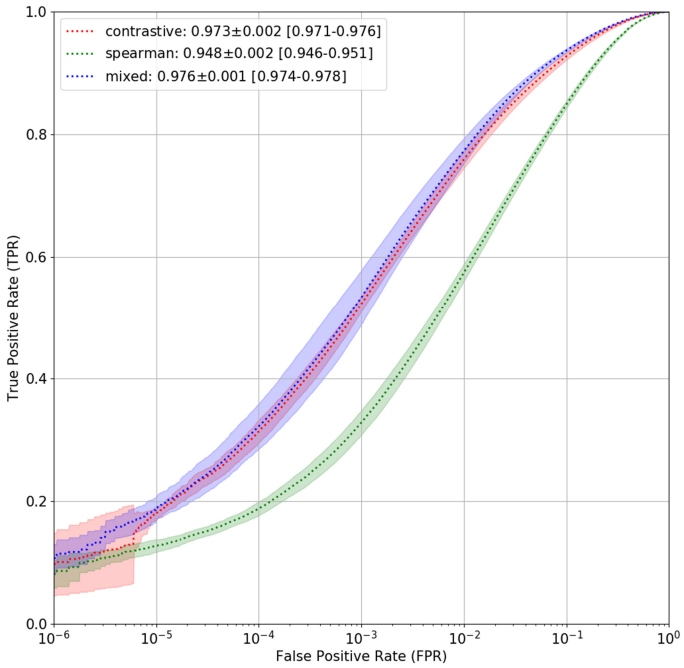


Fig. 7. Results of the Detection Transfer Experiment on SOREL-20M. Classifiers trained on the Contrastive and Mixed embeddings have the highest AU-ROCs of 0.973 ± 0.002 and 0.976 ± 0.001, respectively. Classifiers trained on the Spearman embedding have an AU-ROC of 0.948 ± 0.002.

Results from the malware detection task are shown in Figure 7. Embedding extraction and lightGBM training was performed fives times to obtain error bars. Consistent with our transfer experiments on EMBER, the mixed objective embedding yields the highest AU-ROC, slightly outperforming the contrastive embedding and significantly outperforming the Spearman embedding. For reference, the lightGBM baseline trained on the full 2381-dimensional features has an AU-ROC of 0.981 ± 0.002 [18], a relative average improvement over the top-performing mixed objective model of 0.05%. However, storing the full 2381-dimensional feature vectors requires 74.4 times the amount of storage as that of the 32-dimensional embeddings, indicating that in practice, the top-performing embeddings significantly reduce the storage burden at a slight reduction in net performance.

Our results from the malware attribute labeling task are shown in Table 1. For this task, we fit lightGBM classifiers across each of the malware tags, using 1-hit for each tag as a criterion for presence of the attribute, consistent with Harang and Rudd [18]. Notably, we see a similar pattern with the *Mixed-10* loss on average outperforming the contrastive loss and both losses on average, outperforming the Spearman loss. Note, however, that the mixed loss under-performs the contrastive loss when Spearman performs especially poorly. On average, the results for tagging under-perform the baseline provided with SOREL-20M benchmark, though this is a somewhat invalid comparison, as the attribute baselines from Harang and Rudd [18] utilized a large multi-target network, factoring in number of vendor hits, malicious/benign classification, and simultaneous attribute predictions; thus, some of the performance discrepancy is likely due to limitations of single-target classifiers.

Does the support for each attribute tag within the SOREL train partition affect transfer results? We can assess this using a Kendall–Tau test, another application of the Spearman Ranked Correlation Coefficient. We make two lists; the first containing the rank of the respective support of each attribute (which can be found in Reference [18]) and the second containing the relative order of the *Mixed-10* classifier's performance (AUROC, see

Table 1. Results from the Malware Attribute Transfer Experiment on SOREL-20M

	Contrastive	Spearman	Mixed
Adware	**0.917 ± 0.005**	0.883 ± 0.005	**0.917 ± 0.002**
Crypto Miner	**0.976 ± 0.004**	0.962 ± 0.001	**0.976 ± 0.003**
Downloader	0.832 ± 0.007	0.798 ± 0.005	**0.835 ± 0.004**
Dropper	0.819 ± 0.009	0.773 ± 0.005	**0.824 ± 0.011**
File Infector	0.878 ± 0.003	0.834 ± 0.005	**0.885 ± 0.007**
Flooder	**0.982 ± 0.006**	0.981 ± 0.003	0.979 ± 0.003
Installer	0.957 ± 0.003	0.929 ± 0.002	**0.962 ± 0.002**
Packed	**0.783 ± 0.003**	0.742 ± 0.004	0.779 ± 0.013
Ransomware	0.977 ± 0.003	0.959 ± 0.002	**0.978 ± 0.003**
Spyware	**0.848 ± 0.010**	0.776 ± 0.003	0.846 ± 0.014
Worm	**0.877 ± 0.014**	0.804 ± 0.014	**0.877 ± 0.014**

Mean AU-ROC and AU-ROC standard deviation are reported, with results aggregated over five runs. Best results are shown in bold.

Table 1). We then use the Spearman Rank Correlation Coefficient to conduct a two-tailed hypothesis test, with the null hypothesis that the result is not correlated with support. This yields a coefficient of $r = 0.254$ with a p-value of $p = 0.450$. As this p-value is above any widely used significance level, we cannot reject the null hypothesis, which suggests no strong evidence of relation between the support of each attribute and performance.

5 PRACTICAL DEPLOYMENT CONSIDERATIONS

In prior sections, we maintained an intentional separation between metric learning tasks and downstream transfer tasks in order to assess different metric learning approaches. In practical applications, where we prioritize downstream task performance, we can additionally incorporate these tasks when training the metric learning. Methods of combining downstream transfer tasks with metric learning include pretraining on the transfer task prior to metric learning and incorporating the transfer task as an additional objective while performing metric learning. Similar approaches have been foundational within applied computer vision and facial recognition [16, 35, 37]. In this section, we ask: can enriching our metric embeddings with a detection task lead to superior performance during downstream transfer? To answer this, we repeat a variation of our transfer experiments on EMBER from Section 4.1. These experimental regimes are depicted in Figure 8.

5.1 Enriching Metric Embeddings with Common Transfer Tasks

For our embedding network, we used the same architecture as shown in Figure 3, though this time we vary the size of the final embedding layer to study the effects of different embedding sizes. For pretraining, we use the EMBER 2018 training set with malicious/benign labels, a final dense layer to reduce to 1D, and a sigmoid activation function. We train the initial representation using a binary cross entropy loss. The learned weights are used to initialize the embedding network. We then train the embedding network using a contrastive loss on clusters, as well as an optional malicious/benign loss on the EMBER train set labels. Note that for these experiments, we slightly modify our clustering methodology using VirusTotal's vhash instead of CAPA, with approaches to cluster sampling remaining consistent. This facilitates reproducibility and demonstrates the universality of our approach. We then use the learned embedding network to extract embeddings over the EMBER training set. Finally, we use a LightGBM model to transfer to the specific downstream task. The results of these experiments are shown in Figure 9.

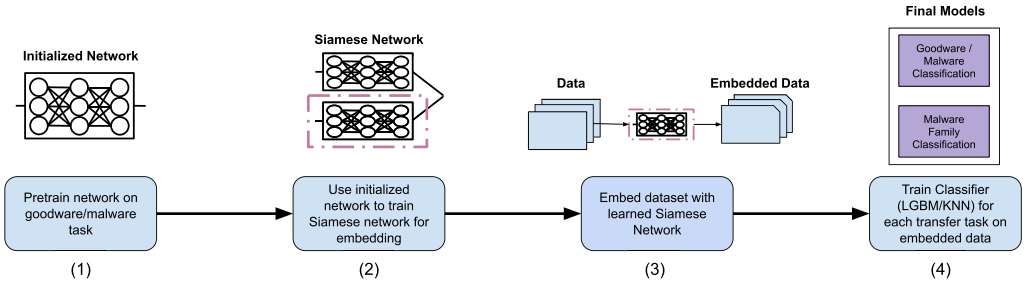
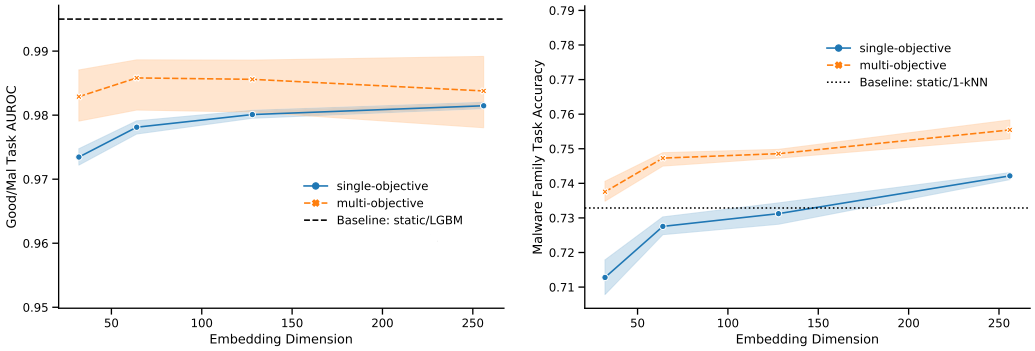


Fig. 8. This figure depicts our training process for enriching our metric embeddings with a detection task. (1) We pretrain a network on goodware/malware classification to initialize effective weights for the next task. (2) We use the pretrained network as the base for an embedding network, which we train using a single-objective (contrastive) loss or multi-objective (contrastive and a binary cross-entropy detection) loss. (3) We then embed a dataset using the learned embedding from (2), represented by the dashed purple box. (4) We perform and evaluate transfer tasks (goodware/malware classification and malware family/type classification).



(a) Goodware/Malware (b) Malware Family

Fig. 9. Transfer Experiments on EMBER. In (a), results of the goodware/malware transfer experiment are reported in terms of the area under the ROC curve (AU-ROC). In (b), results of the malware family transfer experiment are reported in terms of accuracy. For both experiments, classifiers trained on Spearman embeddings under performed those trained on Contrastive embeddings while classifiers trained on embeddings derived from our weighted mixed-objective loss were the top performers.

In both experimental regimes, we tried four different embedding dimensions: 32, 64, 128, and 256. For both tasks, we found that increasing the embedding dimension from 32 to 64 noticeably enhanced performance, regardless of how the embeddings were derived. For goodware/malware classification, when the embedding is derived only from a contrastive loss over clusters, performance increased monotonically with added embedding dimension although gains were gradual beyond 64 dimensions. We also note a slight reduction in variance between runs. When the embeddings are derived from both a contrastive loss over clusters and a classification loss over labels (the multi-objective regime), increasing beyond 64 dimensions decreased performance and increased variance. For the malware family classification task, we also noticed a monotonic increase in performance over both embedding types with embedding dimension, with the steepest performance increase between 32 and 64 dimensions.

The goodware/malware classification experiments showed that both the contrastive (single-objective) embedding and the multi-objective embeddings offered performance superior to that of the neural network used to

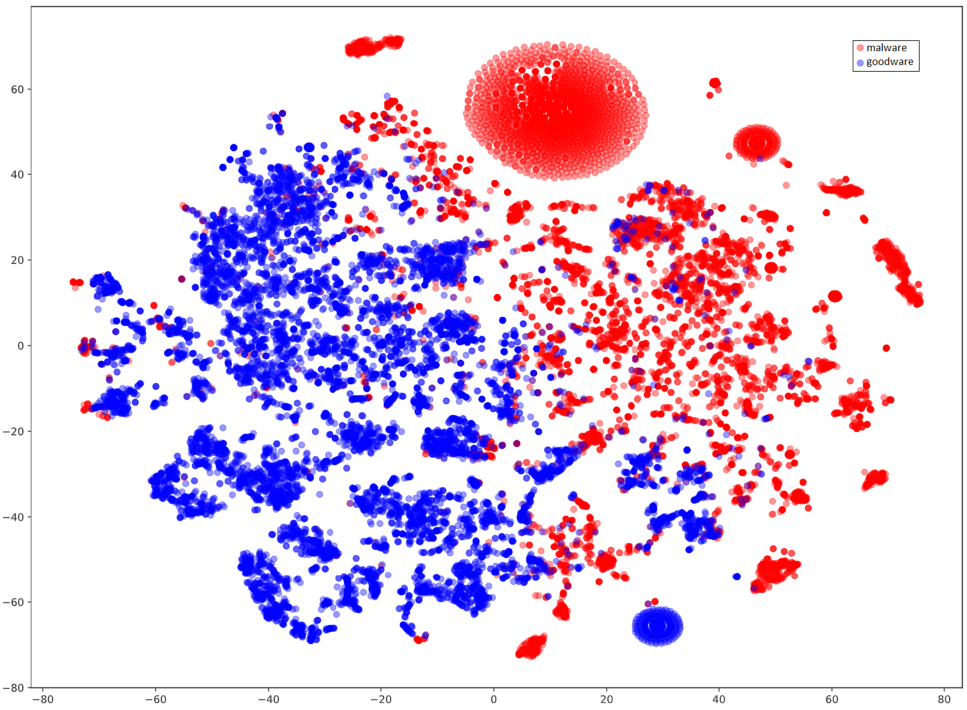


Fig. 10. A qualitative t-SNE visualization of embedded samples from the EMBER test set corresponding to goodware/malware labels. The 64D multi-objective embedding was used to generate these samples. An equivalent number of samples were randomly selected from goodware/malware classes.

initialize the embedding weights, but fell short of a baseline LightGBM classifier trained directly on EMBER features. Meanwhile, in the malware family classification task, we find that regardless of embedding dimension, the multi-objective embedding offers superior performance to the baseline model trained on the much higher-dimensional EMBER feature space, and even the single-objective contrastive embedding outperforms the EMBER feature space with sufficient increase in dimensionality.

5.2 Qualitative Visualization

Our quantitative results from Section 5.1 suggest that our learned embeddings provide separability between classes for a variety of tasks, and in some cases, even provide improved discriminability over the much higher dimension (i.e., 2,381 features) EMBER feature space. In this section, we examine qualitative aspects of these learned embeddings through **t-distributed stochastic neighbor embedding** (**t-SNE**) visualizations [43]. We generate 2D t-SNE visualizations of sample embeddings from the EMBER 2018 test set, sampling over different labels/types. Our base sample embeddings were generated from the multi-dimensional 64-dimensional network described in Section 5.1, since they showed superior performance in the transfer tasks evaluated.

A t-SNE visualization of randomly sampled goodware/malware embeddings is shown in Figure 10, with equal sampling over goodware and malware classes. Note that at a high level, we see general separability between the goodware and malware samples. Based on performance, many of the samples of heterogeneous class that do not appear separated in the 2D projected space are still likely separated in the full 64D embedding space, though these separations would naturally be much more subtle and nonlinear. Specifically of note, in the 2D projected space, we see clear evidence of a decision boundary yet simultaneously a formation of dense clusters

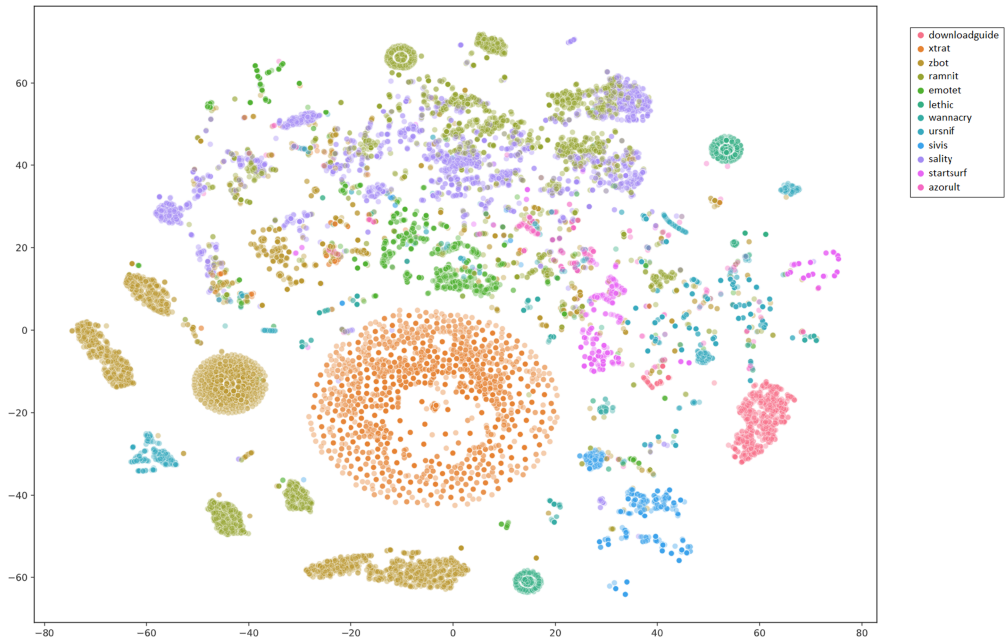


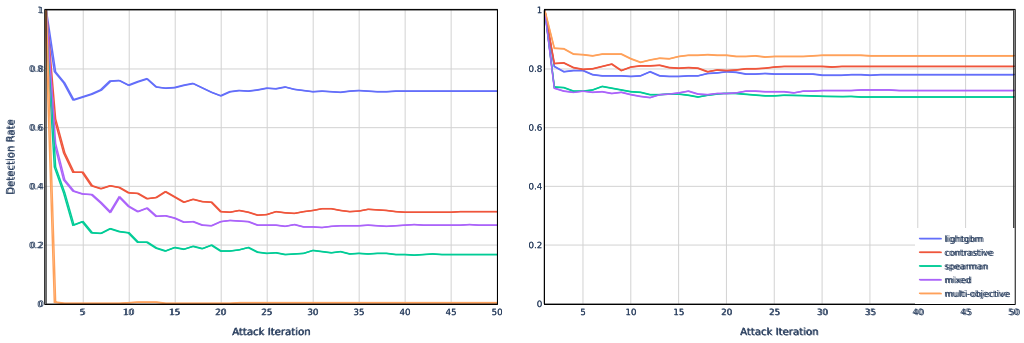
Fig. 11. A qualitative t-SNE visualization of embedded samples from the EMBER test set corresponding to 12 common malware family types.

of samples, indicating that both components of the multi-objective loss appear to have an influence on the embedded space. The contrastive component with respect to clusters creates distinct groups of sample points, while the binary cross-entropy component separates malware from goodware. One can further generally observe that malware samples appear to form larger and more distinct clusters than goodware. Perhaps this is due to greater diversity in content among goodware samples or due to re-use of malicious code within the attacker community.

In Figure 11, we select and visualize samples corresponding to 12 heterogeneous and common malware families from the EMBER test set. Intriguingly, we see that many of these families are distinctly clustered in the t-SNE projected space, even though family information was not used to learn the embedding space. This demonstrates the efficacy of the contrastive loss in learning behavioral clusters, which can be applied toward alternative downstream transfer tasks beyond those directly included in the objective function.

5.3 Analysis of Adversarial Robustness

Given the adversarial nature of our transfer tasks, it is natural for us to investigate the robustness of our learned embeddings to evasion attacks applied in those settings. While many adversarial evasion attacks have been formulated against malware classifiers [1, 25, 41, 42], we focus our analysis on realistic black box attacks leveraging evolutionary learning techniques [11, 12]. These attacks are applicable to both traditional ML models and the neural network models proposed in this article, making them ideal for comparing robustness in a consistent way. Furthermore, since they manipulate the binaries directly, the attacks produce *feasible* adversarial examples that ensure the resultant malware binary is still fully functional. By comparison, white box attacks or those that operate in the feature space require the adversary to gain extraordinary access to the classifier and its gradients, or additional steps to translate feature space manipulations back into the original problem space to create the functional binary [31].



(a) Section Injection Attack (b) Content Shifting Attack

Fig. 12. Detection rates of embedding networks and LightGBM baseline under black-box evasion attack using the GAMMA genetic optimization framework [11, 12].

Specifically, we use Demetrio et al. GAMMA evolutionary framework [11, 12] as implemented in the SecML-Malware framework[1] to examine the robustness of our embedding models and compare it against a baseline LightGBM model trained directly on the 2,381 EMBER features. Two specific attack formulations are presented: Section Injection and Content Shifting. In the Section Injection attack, the adversary adds new section content taken from known-good binaries into the existing malware binary in a way that does not change the other sections. Meanwhile, the Content Shifting attack adds padding from goodware to shift the offset of latter sections within the binary and change their location relative to other structures, such as headers. The evolutionary framework chooses the manipulations evoking the largest reduction in classifier confidence at each attack iteration and uses them as the basis for further manipulation in the next round. In our experiments, we allow up to 50 attack iterations and measure adversary success as the reduction in detection rate from the respective model baselines.

The results of our experiments, shown in Figure 12, demonstrate some interesting trends when examining the success of the GAMMA attack against our embedding models compared to the LightGBM classifier trained directly on the static features. First, it is clear that our embedding models are more sensitive to the Section Injection attack than the LightGBM baseline. The attack reduces the detection rate for our models to between 16.8 and 30.8% for our standard embedding models, and causes complete evasion of the multi-objective model trained on the malware classification task (i.e., detection rate of 0%). At the same time, some embedding models outperform the baseline versus the Content Shifting attack, maintaining a detection rate of up to 84.6% versus 78.2% for the LightGBM model. In both cases, the contrastive-only embedding model performs well compared to the other embedding model variants, while Spearman generally performs poorly.

While a complete analysis of the underlying reasons for these results are beyond the scope of this article, we can hypothesize that the Content Shifting attack has limited impact across all evaluated models because the manipulation does not significantly change the static features underlying the models, or only changes those features that are relatively unimportant in the final model output. The Section Injection attack, on the other hand, affects features that the embedding models rely heavily on for separating samples within the embedding space, such as entropy and number of sections. One interesting observation is that the contrastive-only model remains robust relative to the other embedding models, and that could be due to the coarse nature of the objective itself—focusing on broad notions of positive and negative examples. When losses with additional granularity are used, such as Spearman loss, robustness is noticeably reduced. The binary cross-entropy objective of the

[1] https://github.com/pralab/secml_malware

multi-objective model, meanwhile, interacts with the metric embedding in a way that makes the decision boundary extremely brittle w.r.t. specific features, as observed in Figure 10. Overall, however, it seems like objectives that enforce additional separation among examples within the broader goodware or malware classes open the possibility of introducing weaknesses, perhaps by becoming overly reliant on a small number of features to achieve that separation.

5.4 Cost Analysis

Our embeddings reduce both computational costs and storage costs. Let e_{dim} be the dimension of the embedding and x_{dim} dimension of the original samples' feature vectors. The storage required for the embeddings is thus $\frac{e_{dim}}{x_{dim}}$ of the original, with a reduction in storage of $1 - \frac{e_{dim}}{x_{dim}}$. For EMBER features with a 2301-dimensional feature space and 32-dimensional embeddings, this results in a 98.7% reduction in storage requirement. When training a transfer classifier, costs depend on the type of classifier and training algorithm, but are typically linear complexity (at a minimum) with respect to feature dimension. An exception occurs with pure nearest-neighbor classifiers, where there is no "training" phase, though, in practice, these use some form of data indexing structure (e.g., KD Trees), where the time to fill the structure is proportional to the dimension of the training data. It should also be noted that these types of indexing structures are often more stable with lower-dimensional data. Thus, training a classifier on pre-extracted embeddings offers a training time reduction comparable to a reduction in storage. Depending on data sizes, these storage and computational reductions may lead to significant improvements in overhead and flexibility, as storing and processing embeddings can be conducted on smaller, less expensive, and more flexible hardware, e.g., at the time of this writing, the SOREL transfer experiments can be trained in-memory on a laptop when using 32-dimensional embeddings. The original SOREL dataset using EMBER features takes up more than 300 GB of RAM.

6 CONCLUSION

In this article, we introduced two different approaches to enrich metric embeddings with static disassembly capabilities information and performed evaluations thereof on multiple downstream tasks. These approaches, outlined in Section 3, consist of a fine-grained Spearman embedding approach and a coarse-grained contrastive embedding approach. In the vast majority of our experiments in Section 4, the coarse-grained contrastive approach exhibited superior performance to the finer-grained Spearman approach. In some respects, this is not surprising, as contrastive loss inherently forces separability in a way that the Spearman loss does not. An in-depth examination of similarity distributions and adoption of additional similarity measures other than Jaccard similarity could be helpful in improving the Spearman embedding. Consistent with other literature in the ML security/applied ML space, we found that combining both Spearman and contrastive embedding losses generally improved performance as did balancing loss contributions to a similar order of magnitude [34–36]. Furthermore, adding task-specific objectives greatly improved performance on multiple downstream tasks, but at the cost of catastrophic loss of robustness to evasion.

When trained from scratch, using no transfer learning or auxiliary downstream task losses, our embeddings performed comparably to classifiers trained on raw features for certain tasks, but did not work so well for others. Among a variety of factors, this may be due to including semantic information inherent to the CAPA embeddings, over/under-fitting, and hyperparameter selection. While we were not able to outperform our baselines using transfer from "from-scratch" metric learnt embeddings alone, we were able to do so on the malware family classification task in Section 5.1, using heterogeneous labeling in conjunction with metric learning in a manner very similar to approaches previously explored in computer vision literature [16, 37].

Generally, we surmise that improving the performance of metric embeddings for additional tasks is trivially feasible by utilizing additional training objectives. These may include malicious/benign labels, malware attribute tags, MITRE ATT&CK tactics (notably, CAPA outputs these as well as attributes) and additional metadata.

Moreover, examining how embedding performance scales when training on larger more heterogeneous groups of samples and evaluating on substantially concept-drifted data could offer further insight into embedding performance and design (e.g., Reference [47]).

We could utilize our embeddings to address a multitude of other malware analysis use-cases, including attribution of malware to specific APT groups and threat hunting. Filenames, SHA values, and other telemetry can be found in open source threat reports from a multitude of vendors, as well as tactics, strategies, and **indicators of compromise** (**IOCs**). Using just the SHAs along with a visualization method (e.g., UMAP, T-SNE), one could inspect other nearby malware samples, or even fine-tune classifiers to predict APT-group membership.

Notably, a low-dimensional embedding which performs well for a variety of classification/information retrieval tasks could yield significant computational and storage savings over utilizing raw features or binaries. Such embeddings could be utilized in both academic contexts, where compute resources are often limited or in commercial contexts for rapid prototyping. As a reference, the SOREL-20M dataset is an order of magnitude smaller than industry datasets which are typically used to train commercial PE malware detectors, yet it still comes with a warning about potentially incurring bandwidth fees or exhausting disk space. Even in featurized format as 32-bit floating point, the SOREL-20M dataset requires 172 GB of storage. Using the embeddings introduced in this paper, this can be compressed to roughly 2.3 GB, which is small enough to fit in memory, even for most laptops.

REFERENCES

[1] Hyrum S. Anderson, Anant Kharkar, Bobby Filar, David Evans, and Phil Roth. 2018. Learning to evade static PE machine learning malware models via reinforcement learning. *arXiv preprint arXiv:1801.08917* (2018).

[2] Hyrum S. Anderson and Phil Roth. 2018. EMBER: An open dataset for training static pe malware machine learning models. *arXiv preprint arXiv:1804.04637* (2018).

[3] W. Ballenthin and M. Raabe. 2020. capa: Automatically identify malware capabilities. (2020). Retrieved from https://www.mandiant.com/resources/capa-automatically-identify-malware-capabilities. Accessed: 2022-08-05.

[4] Mathieu Blondel, Olivier Teboul, Quentin Berthet, and Josip Djolonga. 2020. Fast differentiable sorting and ranking. In *Proceedings of the International Conference on Machine Learning*. PMLR, 950–959.

[5] Frank Breitinger, Knut Petter Astebol, Harald Baier, and Christoph Busch. 2013. mvHash-B—A new approach for similarity preserving hashing. In *Proceedings of the 2013 Seventh International Conference on IT Security Incident Management and IT Forensics (IMF'13)*. IEEE Computer Society, 33–44. DOI: https://doi.org/10.1109/IMF.2013.18

[6] Frank Breitinger and Harald Baier. 2013. Similarity preserving hashing: Eligible properties and a new algorithm MRSH-v2. In *Proceedings of the Digital Forensics and Cyber Crime*. 167–182. DOI: https://doi.org/10.1007/978-3-642-39891-9_11

[7] Frank Breitinger, Harald Baier, and Douglas White. 2014. On the database lookup problem of approximate matching. *Digital Investigation* 11, S1 (May 2014), S1–S9. https://www.sciencedirect.com/science/article/pii/S1742287614000061

[8] Frank Breitinger, Christian Rathgeb, and Harald Baier. 2014. An efficient similarity digests database lookup—A logarithmic divide & conquer approach. *The Journal of Digital Forensics, Security and Law (JDFSL)* 9, 2 (2014), 155–166. DOI: http://ojs.jdfsl.org/index.php/jdfsl/article/view/276

[9] Mahinthan Chandramohan, Yinxing Xue, Zhengzi Xu, Yang Liu, Chia Yuan Cho, and Hee Beng Kuan Tan. 2016. BinGo: Cross-architecture cross-OS Binary Search. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE 2016)*. ACM, New York, NY, 678–689. DOI: https://doi.org/10.1145/2950290.2950350

[10] Xiao Chen, Zhengwei Jiang, Shuwei Wang, Rongqi Jing, Chen Ling, and Qiuyun Wang. 2022. Malware detected and tell me why: An verifiable malware detection model with graph metric learning. In *Proceedings of the Science of Cyber Security: 4th International Conference, SciSec 2022, Matsue, Japan, August 10–12, 2022, Revised Selected Papers*. Springer, 302–314.

[11] Luca Demetrio, Battista Biggio, Giovanni Lagorio, Fabio Roli, and Alessandro Armando. 2021. Functionality-preserving black-box optimization of adversarial windows malware. *IEEE Transactions on Information Forensics and Security* 16 (2021), 3469–3478.

[12] Luca Demetrio, Scott E. Coull, Battista Biggio, Giovanni Lagorio, Alessandro Armando, and Fabio Roli. 2021. Adversarial exemples: A survey and experimental evaluation of practical attacks on machine learning for windows malware detection. *ACM Transactions on Privacy and Security (TOPS)* 24, 4 (2021), 1–31.

[13] Mirabelle Dib, Sadegh Torabi, Elias Bou-Harb, Nizar Bouguila, and Chadi Assi. 2022. EVOLIoT: A self-supervised contrastive learning framework for detecting and characterizing evolving IoT malware variants. In *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*. 452–466.

[14] Steven H. H. Ding, Benjamin C. M. Fung, and Philippe Charland. 2016. Kam1N0: MapReduce-based assembly clone search for reverse engineering. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'16)*. ACM, New York, NY, 461–470. DOI: https://doi.org/10.1145/2939672.2939719

[15] Steven H. H. Ding, Benjamin C. M. Fung, and Philippe Charland. 2019. Asm2Vec: Boosting static representation robustness for binary clone search against code obfuscation and compiler optimization. In *Proceedings of the 2019 IEEE Symposium on Security and Privacy (SP)*. DOI: https://doi.org/10.1109/SP.2019.00003

[16] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. 2014. Decaf: A deep convolutional activation feature for generic visual recognition. In *Proceedings of the International Conference on Machine Learning*. PMLR, 647–655.

[17] Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics*. JMLR Workshop and Conference Proceedings, 249–256.

[18] Richard Harang and Ethan M. Rudd. 2020. SOREL-20M: A large scale benchmark dataset for malicious PE detection. *arXiv preprint arXiv:2012.07634* (2020).

[19] Elad Hoffer and Nir Ailon. 2015. Deep metric learning using triplet network. In *Proceedings of the International Workshop on Similarity-based Pattern Recognition*. Springer, 84–92.

[20] Martin Jurecek, Olha Jurecková, and Róbert Lórencz. 2021. Improving classification of malware families using learning a distance metric. In *Proceedings of the ICISSP*. 643–652.

[21] Martin Jurecek and Róbert Lórencz. 2020. Distance metric learning using particle swarm optimization to improve static malware detection. In *Proceedings of the ICISSP*. 725–732.

[22] Martin Jureček and Róbert Lórencz. 2021. Application of distance metric learning to automated malware detection. *IEEE Access* 9, 1 (2021), 96151–96165. https://ieeexplore.ieee.org/abstract/document/9469874

[23] Mahmut Kaya and Hasan Şakir Bilge. 2019. Deep metric learning: A survey. *Symmetry* 11, 9 (2019), 1066.

[24] Gregory Koch, Richard Zemel, Ruslan Salakhutdinov, et al. 2015. Siamese neural networks for one-shot image recognition. In *Proceedings of the ICML Deep Learning Workshop*, Vol. 2. Lille, 0.

[25] Bojan Kolosnjaji, Ambra Demontis, Battista Biggio, Davide Maiorca, Giorgio Giacinto, Claudia Eckert, and Fabio Roli. 2018. Adversarial malware binaries: Evading deep learning for malware detection in executables. In *Proceedings of the 2018 26th European Signal Processing Conference (EUSIPCO)*. IEEE, 533–537.

[26] Xuezixiang Li, Yu Qu, and Heng Yin. 2021. PalmTree: Learning an assembly language model for instruction embedding. In *Proceedings of the CCS*.

[27] David Lillis, Frank Breitinger, and Mark Scanlon. 2017. Expediting MRSH-v2 approximate matching with hierarchical bloom filter trees. In *Proceedings of the 9th EAI International Conference on Digital Forensics and Cyber Crime (ICDF2C 2017)*. Springer.

[28] Chen Liu, Bo Li, Jun Zhao, Ziyang Zhen, Xudong Liu, and Qunshi Zhang. 2022. FewM-HGCL: Few-shot malware variants detection via heterogeneous graph contrastive learning. *IEEE Transactions on Dependable and Secure Computing* 1 (2022), 1–18. https://ieeexplore.ieee.org/document/9928211/citations#citations

[29] Luca Massarelli, Giuseppe Antonio Di Luna, Fabio Petroni, Leonardo Querzoni, and Roberto Baldoni. 2019. SAFE: Self-attentive function embeddings for binary similarity. In *Proceedings of the Detection of Intrusions and Malware, and Vulnerability Assessment*. 309–329.

[30] Jonathan Oliver, Chun Cheng, and Yanggui Chen. 2013. TLSH—A locality sensitive hash. In *Proceedings of the 2013 Fourth Cybercrime and Trustworthy Computing Workshop*. IEEE, 7–13. DOI: https://doi.org/10.1109/CTC.2013.9

[31] Fabio Pierazzi, Feargus Pendlebury, Jacopo Cortellazzi, and Lorenzo Cavallaro. 2020. Intriguing properties of adversarial ml attacks in the problem space. In *Proceedings of the 2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1332–1349.

[32] Edward Raff and Charles Nicholas. 2017. Malware classification and class imbalance via stochastic hashed LZJD. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security (AISec'17)*. ACM, New York, NY, 111–120. DOI: https://doi.org/10.1145/3128572.3140446

[33] Edward Raff and Charles K. Nicholas. 2018. Lempel-Ziv jaccard distance, an effective alternative to ssdeep and sdhash. *Digital Investigation* 24, 1 (2018), 34–49. DOI: https://doi.org/10.1016/j.diin.2017.12.004

[34] Ethan M. Rudd, Felipe N. Ducau, Cody Wild, Konstantin Berlin, and Richard Harang. 2019. {ALOHA}: Auxiliary loss optimization for hypothesis augmentation. In *Proceedings of the 28th USENIX Security Symposium (USENIX Security 19)*. 303–320.

[35] Ethan M. Rudd, Manuel Günther, and Terrance E. Boult. 2016. Moon: A mixed objective optimization network for the recognition of facial attributes. In *Proceedings of the European Conference on Computer Vision*. Springer, 19–35.

[36] Ethan M. Rudd, Mohammad Saidur Rahman, and Philip Tully. 2022. Transformers for end-to-end InfoSec tasks: A feasibility study. In *Proceedings of the 1st Workshop on Robust Malware Analysis*. 21–31.

[37] Florian Schroff, Dmitry Kalenichenko, and James Philbin. 2015. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 815–823.

[38] David Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, and Michael Young. 2014. Machine learning: The high interest credit card of technical debt. (2014).

[39] Marcos Sebastián, Richard Rivera, Platon Kotzias, and Juan Caballero. 2016. Avclass: A tool for massive malware labeling. In *Proceedings of the International Symposium on Research in Attacks, Intrusions, and Defenses*. Springer, 230–253.

[40] Eui Chul Richard Shin, Dawn Song, and Reza Moazzezi. 2015. Recognizing functions in binaries with neural networks. In *Proceedings of the 24th USENIX Security Symposium (USENIX Security 15)*. 611–626.
[41] Wei Song, Xuezixiang Li, Sadia Afroz, Deepali Garg, Dmitry Kuznetsov, and Heng Yin. 2022. MAB-Malware: A reinforcement learning framework for blackbox generation of adversarial malware. In *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security* (Nagasaki, Japan). Association for Computing Machinery, 990–1003.
[42] Octavian Suciu, Scott E. Coull, and Jeffrey Johns. 2019. Exploring adversarial examples in malware detection. In *Proceedings of the 2019 IEEE Security and Privacy Workshops (SPW)*. IEEE, 8–14.
[43] Laurens Van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of Machine Learning Research* 9, 11 (2008), 2579–2605.
[44] VirusTotal 2022. VirusTotal—Stats. Retrieved from https://www.virustotal.com/gui/stats. Accessed: 2022-08-04.
[45] Christian Winter, Markus Schneider, and York Yannikos. 2013. F2S2: Fast forensic similarity search through indexing piecewise hash signatures. *Digital Investigation* 10, 4 (Dec 2013), 361–371. DOI: https://doi.org/10.1016/j.diin.2013.08.003
[46] Xiaojun Xu, Chang Liu, Qian Feng, Heng Yin, Le Song, and Dawn Song. 2017. Neural network-based graph embedding for cross-platform binary code similarity detection. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 363–376.
[47] Limin Yang, Arridhana Ciptadi, Ihar Laziuk, Ali Ahmadzadeh, and Gang Wang. 2021. BODMAS: An open dataset for learning based temporal analysis of PE malware. In *Proceedings of the 4th Deep Learning and Security Workshop*.
[48] Shouguo Yang, Long Cheng, Yicheng Zeng, Zhe Lang, Hongsong Zhu, and Zhiqiang Shi. 2021. Asteria: Deep learning-based ast-encoding for cross-platform binary code similarity detection. In *Proceedings of the 2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 224–236. DOI: https://doi.org/10.1109/DSN48987.2021.00036
[49] Shaojie Yang, Yongjun Wang, Haoran Xu, Fangliang Xu, and Mantun Chen. 2022. An android malware detection and classification approach based on contrastive lerning. *Computers & Security* 123, 1 (2022), 102915. https://www.sciencedirect.com/science/article/pii/S016740482200308X
[50] Shuofei Zhu, Jianjun Shi, Limin Yang, Boqin Qin, Ziyi Zhang, Linhai Song, and Gang Wang. 2020. Measuring and modeling the label dynamics of online anti-malware engines. In *Proceedings of the 29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, 2361–2378. Retrieved from https://www.usenix.org/conference/usenixsecurity20/presentation/zhu

Received 1 December 2022; revised 25 May 2023; accepted 17 July 2023