Towards a Utopia of Dataset Sharing: A Case Study on Machine Learning-based Malware Detection Algorithms

Ping-Jui Chuang
m.c@nycu.edu.tw
National Yang Ming Chiao Tung University
Hsinchu, Taiwan

Chih-Fan Hsu
hsuchihfan@gmail.com
National Yang Ming Chiao Tung University
Hsinchu, Taiwan

Yun-Tien Chu
yungtien.cs09@nycu.edu.tw
National Yang Ming Chiao Tung University
Hsinchu, Taiwan

Szu-Chun Huang
schuang.cs09g@nctu.edu.tw
National Yang Ming Chiao Tung University
Hsinchu, Taiwan

Chun-Ying Huang
chuang@cs.nctu.edu.tw
National Yang Ming Chiao Tung University
Hsinchu, Taiwan

ABSTRACT

Working with a high-quality (complete and up-to-date) dataset is the key to building a good machine learning model, especially in security research areas. However, it is not easy to collect a good quality dataset for security research communities because of the sensitive property of most security datasets. We believe that having more contributors to share up-to-date samples would increase the quality of datasets. Therefore, this study aims to increase security dataset sharing for research communities by eliminating possible information leakage. We propose a dataset sharing model and the core algorithm, FeatureTransformer, which guarantees no sensitive information leakage from a shared dataset. FeatureTransformer transforms extracted raw features into intermediate features that conceal sensitive information. Meanwhile, models built from transformed features maintain similar performance compared to models built from the original raw features. We show the effectiveness of our model by evaluating FeatureTransformer with typical malware classification problems using (1) traditional machine learning classifiers and (2) neural network-based classifiers. The experiment results show that the models trained with transformed features merely suffer from 2.56% and 1.48% accuracy degradation on the investigated problems. It indicates that models validated by datasets processed by FeatureTransformer work well with the original raw (untransformed) datasets. We believe that our privacy-preserving model can stimulate dataset sharing and advance the development of machine learning approaches in solving security problems.

ASIA CCS '22, May 30–June 3, 2022, Nagasaki, Japan

ACM ISBN 978-1-4503-9140-5/22/05…$15.00
https://doi.org/10.1145/3488932.3497763

CCS CONCEPTS

• **Security and privacy** → **Domain-specific security and privacy architectures**; **Usability in security and privacy**; *Data anonymization and sanitization*; • **Computing methodologies** → *Machine learning approaches.*

KEYWORDS

Dataset Sharing, Machine Learning, Malware Classification, Reproducible Research

ACM Reference Format:
Ping-Jui Chuang, Chih-Fan Hsu, Yun-Tien Chu, Szu-Chun Huang, and Chun-Ying Huang. 2022. Towards a Utopia of Dataset Sharing: A Case Study on Machine Learning-based Malware Detection Algorithms. In *Proceedings of the 2022 ACM Asia Conference on Computer and Communications Security (ASIA CCS '22), May 30–June 3, 2022, Nagasaki, Japan.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3488932.3497763

1 INTRODUCTION

With the rapid development of machine learning (ML) techniques, researchers have started to solve research problems by creating and using modern ML models. Datasets are critical for researchers to conduct ML research works. It is well-known that WordNet [36] and ImageNet [15] datasets have successfully boosted the research in natural language processing and image classification algorithms, respectively. The wide use of the datasets mentioned above also proves that the key to accelerating ML research and building successful ML models is the availability and completeness of the dataset. Machine learning techniques have also been used in solving security problems not only limited to fields such as signal processing [42], natural language processing [40, 47], and image classification [15, 27]. Therefore, it would be foreseen that working with a high-quality (sufficient diversity, up-to-date, correct labeling, and high availability) dataset is a critical step towards successful machine learning-based security research works.

Although working with a high-quality dataset is the key to success, it is not easy to have it for research communities. The difficulty is mainly because the nature of samples in security datasets is different from that of samples collected in the human world, such as WordNet and ImageNet. Take the example of malware samples used

in malware detection and classification research works. First, the sample sources are limited. While almost everyone can contribute samples like sentences or pictures, *malware samples can only be provided by professionals*. A contributor must confirm that a shared sample is an actual malware and label its class (benign/malicious) or family correctly. Second, *it is not easy to perform augmentation against existing samples*. Sample augmentation is a common technique to increase the number of samples in a dataset. However, it does not work for malware samples. An augmented sample could be broken and therefore loses essential features that can be used for malware detection and classification. Third, *security datasets have to be fresh*. A well-trained model based on malware samples collected three years ago could be useless for detecting malware samples nowadays. Last and most importantly, *there could be privacy issues when sharing a malware sample*. Some APT malware samples embed sensitive information in their binary and could leak knowledge about victims. For example, the ShadowHammer [29] APT malware embeds encrypted MAC addresses of targets in the binary and can be extracted and decrypted by skillful hackers. While the first three differences already set up a high burden for building a high-quality dataset, the last one could further lower the willingness of dataset sharing.

Existing publicly available security datasets employ different workarounds to avoid the difficulties mentioned above. For example, the Ember dataset [8] selects only statistical features and printable text-based features. Alternatively, the Microsoft malware classification challenge (BIG 2015) [43] hosted on Kaggle shares analyzer-generated metadata and most parts of binary content of malware samples. However, it removes Portable Executable (PE) headers from the binary content and performs proprietary anonymize operations against the shared content. Although the approaches mentioned above are a good fit for the shared dataset, they cannot be generalized for all datasets and algorithms and limit exploring alternative features for malware detection and classification. For example, a binary classification algorithm may consider working with raw byte sequences extracted from malware samples [25, 41, 55].

In this study, we aim to increase the public willingness of dataset sharing for security research communities. We believe a transformation approach to transform extracted raw features into intermediate features that conceal sensitive information is required to achieve this goal. In addition to sensitive information concealing, a transformation approach cannot degrade an algorithm that performs training and detection based on the transformed features. Our contribution is three-fold. First, we describe our proposed dataset sharing model that can be used for both research communities and security industries. Second, we proposed FeatureTransformer, which is the core algorithm used in the dataset sharing model. Third, we perform in-depth evaluations against our proposed FeatureTransformer to show its performance and effectiveness. To be more specific, we answer the following research questions (RQs) to validate our research.

- *RQ1*. Does FeatureTransformer Really Work? (Sec. 5.2)
- *RQ2*. Can we use the same approach for handling text-based and numeric features? (Sec. 5.3)
- *RQ3*. What are the impacts of algorithm parameters for FeatureTransformer? (Sec. 5.4)
- *RQ4*. Is that possible to have access control to limit the audiences of shared samples? (Sec. 5.5)
- *RQ5*. What are the impacts of transformation algorithms for FeatureTransformer? (Sec. 5.6)
- *RQ6*. What is the performance of FeatureTransformer when applying it to existing malware detection algorithms? (Sec. 5.7)
- *RQ7*. Can we apply FeatureTransformer to generic machine learning algorithms? (Sec. 5.8)

The rest of this paper is organized as follows. We present essential backgrounds and related works in Section 2. Our proposed dataset sharing model is illustrated in Section 3. The core algorithm, FeatureTransformer, is introduced in Section 4. Section 5 presents how the proposed algorithms are evaluated and answer the research questions. A concluding remark is given in Section 6. To simplify the discussions, we use malware detection as the main scenario throughout this paper. Nevertheless, the proposed dataset sharing model and algorithms can be easily applied to other scenarios.

2 RELATED WORKS

This section introduces the development of machine learning techniques on malware detection, current publicly available malware repositories, and privacy-preserving approaches.

2.1 Machine Learning-based on Malware Detection

Using machine learning (ML) techniques to detect malware is considered practical [9, 35, 39] because ML algorithms can effectively learn the behaviors or patterns from samples. A well-trained ML algorithm would efficiently and automatically detect malware with representative samples (or features obtained from static or dynamic analysis from samples). Several research works have been done to solve security problems with machine learning approaches. Zhao et al. [60] extract specific sensitive Application Programming Interface (API) calls from malware to support detecting unknown Android PacKages (APKs) based on the decision tree and k-nearest neighbor algorithms. By transforming the API calls invoked during execution into image-like sparse matrices, D'Angelo et al. [18] proposed an ML-based method to detect malware on Android-based devices. They extract representative features from the sparse matrices and feed them to a Neural Network (NN)-based classifier. Gibert et al. [20] detect malware based on an NN-based multi-model architecture that comprises three sub-models for obtaining information from API calls, opcode characteristics, and raw byte, respectively. Feng et al. [19] presented a method for classifying malicious Android app based on the NN-based model. They target features including manifest properties, API calls, and opcode sequences. Yin et al. [56] use a structured Heterogeneous Information Network (HIN) to detect malicious Portable Executable (PE) files. The input of HIN covers the information of PE header, Dynamic-Link Libraries (DLLs), API call sequences, and opcode sequences. Kapoor et al. [28] extract features from app's permissions. Then, they train several ML algorithms based on the features for classifying malicious Android applications. Surendran et al. [50] proposed a malware detection method based on a Tree Augmented Naive Bayes (TAN)-based hybrid mechanism. Three logistic regression classifiers corresponding

to API calls, permission, and system calls are modeled. Then, the TAN framework estimates the inter-dependency of the classifier outputs to detect malware. Ajeena Beegom et al. [7] proposed a method for detecting malware based on a Support Vector Machine (SVM) classifier with integrated static features including permissions, API calls, and opcodes. Nauman et al. [37] present several NN-based models to detect Android malware based on multiple features, including permissions usage, API calls, and intent-related network activities.

Although the development of ML models on malware detection achieves significant success, a representative malware dataset is still required for reproducible research works to compare the performance of different models. However, malware datasets are either private or limited because of the difficulties of collecting samples.

2.2 Public Available Malware Repositories and Datasets

Several malware datasets are available on the Internet. Generally, the available datasets share either the original samples or extract features from private samples. VirusTotal [1], a company that provides malware scanning and searching services, maintains a malware repository that contains more than 2.4B files of various types. The online repository is widely used for numerous malware-related researches from different perspectives, such as malware hunting [57], packing [16, 34], and dynamic analysis [30]. Zhu et al. [61] surveyed 115 publications that adopt anti-malware engines provided by VirusTotal. By manually verifying and analyzing files collected from the engines over a year, they validated the benefits of threshold-based labeling methods. VirusShare [24], another online malware repository, collects about 38M live malware. Users can upload malware samples onto the online repository. Afterward, VirusShare reports virus scanning results for the uploaded file. Users can also search and download scanned files for further research, such as analyzing or detecting Personal Computer (PC) and Android malware [11, 52, 59]. Sharing live samples is helpful for researching but is risky to the public. However, users would require professional knowledge to handle the samples safely and prevent their devices from being infected and damaged. Furthermore, shared malware samples might carry sensitive information, such as IP address, MAC address, or personal information. This sensitive information should be well protected.

Other than sharing live malware, some repositories share representative features extracted from malware samples, for example, the EMBER dataset [8] (1.1M samples) and the SoReL-20M dataset [21] (20M samples). Although the shared features suffer from a certain degree of information loss compared to raw samples, downloading extracted features avoids the risk of being infected. However, the risk of leaking private information may still not be completely avoided because adversaries may attempt to derive sensitive information from features.

2.3 Privacy Preserving

To avoid accidentally leaking private information, making sharers secure their samples without uploading them to the Internet seems a solution. Several novel techniques are proposed to improve model training without uploading samples or extracted features—for example, differential privacy [17] and federated learning [10]. Hsu et al. [23] proposed an image transformation method, HE-SIFT (homomorphic encryption-based secure SIFT), to transform raw 2D images to the corresponding ciphered images to preserve the raw images' privacy information. The ciphered images are visually unrecognizable to humans. The experiment results show that the detected SIFT features between the raw and corresponding ciphered images achieve spatial consistency. Abadi et al. [4] propose an algorithm that combines differential privacy and an NN-based algorithm to improve the computational efficiency of network training. The authors test their algorithm on the MNIST and CIFAR-10 datasets and prove that the algorithm not only reduces the computational cost and improves the training efficiency and but also improves detection accuracy.

Nowadays, federated learning [58] has become increasingly popular. Federated learning is a set of methods that allows multiple clients (or nodes) to collaborate to train machine learning models without exchanging samples. In this case, data owners could keep their data in the local, which significantly increases data privacy. Several implementations [31, 33, 33, 44] have employed federated learning in their research works. Among these works, Lin et al. [32] use federated learning on the malware classification problem. The authors integrate a federated learning method with LSTM (Long short-term memory) model and compare its performance against a traditional SVM (support vector machine) model. The results show the superior of the federated learning approach on the detection accuracy while the size of training samples is small (1,000 training samples in their experiment). Although federated learning methods are attractive, a few challenges should be further addressed, (i) the current approaches suffer from inevitable performance degradation, such as detection accuracy in the detection problem; (ii) federated learning approaches generally require longer training time than traditional centralized approaches because the clients have to exchange some temporary information during training, such as the intermediate model weights or model outputs; and (iii) data owners must be online and constantly spend additional computational resources and network bandwidth to train and transmit the temporary information. Federated learning is beneficial to deploying distributed machine learning models, but it could be an overkill for building centralized and reproducible experimental datasets.

3 MOTIVATION

3.1 Impacts of Dataset Quality

We first show how dataset quality affects the performance of a machine learning model. Suppose the number of samples in a class is limited or even not available. In that case, a classification model trained from the samples may not achieve acceptable performance to detect that class due to insufficient information about the missing class. Researchers have attempted to propose few-shot learning [54] and zero-shot learning [53] to augment samples or perform transfer learning with limited information. The intuition of the few-shot and zero-shot learning is improving the adaptation ability of the models to adapt to the scarce classes rapidly. However, few-shot and zero-shot learning work under the assumption that the model builder has prior knowledge about the similarity of samples, the

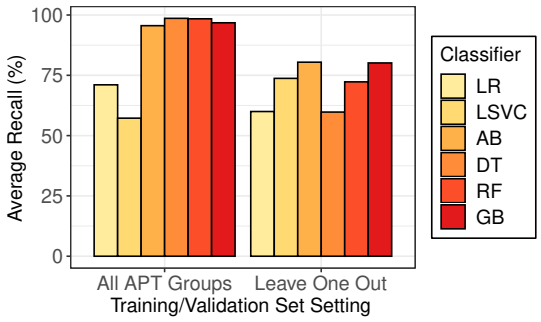


Figure 1: The average detection recall under different settings of training and validation sets.

distribution, and the properties. Although few-shot and zero-shot learning approaches have obtained great success in areas of image classification, these approaches inevitably suffer from a certain level of accuracy degradation, which could not be acceptable for solving security problems. For example, incorrectly classifying a variant of a type of ransomware as benign is serious because it could bring data and monetary loss to the users.

We conduct a pilot study to investigate the impact of the imbalanced data in the malware detection problem. We attempted to build a classification model for detecting APT malware based on the samples from a publicly available APT dataset [3]. The APT dataset contains 3,594 malware samples with twelve different malware groups. We compared the results from two experiments of different settings to investigate the impacts when a model has no information for a missing malware group. We randomly sample 70% of malware in each group as the training set and use the remaining 30% as the validation set in the first setting. We then use the same ratio for sample selection in the second setting but leave one malware group out of the training set. The validation set remains the same. For each setting, we also randomly select an equivalent number of benign samples from the EMBER dataset [8]. We preprocess the malware samples to the trainable feature by the method proposed by Anderson and Roth [8]. We select six popular binary classification algorithms to build classification models. The algorithms include Linear Regression (LR), Linear Support Vector Classifier (LSVC), AdaBoost (AB), Decision Tree (DT), Random Forest (RF), and Gradient Boosting (GB). Figure 1 shows the average detection result. We observe that the detection recall significantly decreased when one APT group is entirely unseen by the classifiers. The results show the importance of sharing malware samples. Once a detection model covers samples from an APT group (or a family), the model is more likely to identify the malware that belongs to the trained family. Users may only rely on a detection model to protect their property if that model is built from sufficient sample coverage.

3.2 Privacy-Preserved Dataset Sharing

We believe the first priority for building a healthy sample-sharing community is to guarantee the privacy of shared samples. Our observations have shown that existing research works reflect the so-called isolated data island issue. Different research works analyze or evaluate their approaches with private datasets. Hence, the results of these research works may not be reproducible by others. The accumulation of non-reproducible methods could significantly impede the development of a field because the experiment results and conclusions cannot be effectively verified and improved. In order to mitigate this situation, an open-sourced platform, Papers With Code [2], is established to encourage researchers and developers to share their samples and codes. It is an excellent start to encourage researchers to share and would lower the barrier for followed researches.

However, the willingness to share malware samples could be degraded because (i) the public may not know how to safely store malware samples without taking the risk of being infected; (ii) the samples are the private asset for certain facilities that takes a tremendous effort for the sample collection; and (iii) shared samples may leak sensitive data such as the information of victims. Therefore, we propose a generalized privacy-preserved sample sharing model to address the issues mentioned above and improve the willingness for sample sharing. The principle of our proposed model is to transform extracted original features into different characters or vector spaces, depending on the type of the features. The objectives of our proposed privacy-preserved sample sharing are as follows:

(1) *Privacy-preserving*: The shared features in a dataset cannot leak sensitive information to the public, even if an entire binary sample file is considered a feature.
(2) *Consistent performance*: The classification models built from the transformed dataset should have a similar detection accuracy and recall rate to those built from original features. With this property, a model built based on transformed features can effectively detect other transformed samples. Furthermore, it can also faithfully reflect the performance if a model is built and evaluated with untransformed samples.
(3) *Access control*: Our proposed model has two different access privileges, namely complete and limited. A sample contributor can control who has complete access privileges to the shared dataset. Users with a limited access privilege can only use the shared dataset to build and test a built model against only samples in the shared dataset. In contrast, users with a complete access privilege can additionally add new samples to the shared dataset and use a built model to detect any arbitrary samples, even if a sample is initially not in the dataset.

Figure 2 shows the application scenario of our privacy-preserved dataset sharing model. In the scenario, the samples s and original features f are private data owned by the sample contributors and the model users. The parameters p (depicted as keys in the figure) used for performing feature transformation are only accessible to users who work on the same repository. The rest data and components, including the algorithms and public repositories, are known to the public in the figure. We mention here that $X(\cdot)$, $T(\cdot)$, and $V(\cdot)$ functions indicate the process of feature extraction, feature transformation, and feature vectorization, respectively. The application scenario can be divided into three phases, i.e., the sample-sharing phase by sample contributors, the model-building phase by security

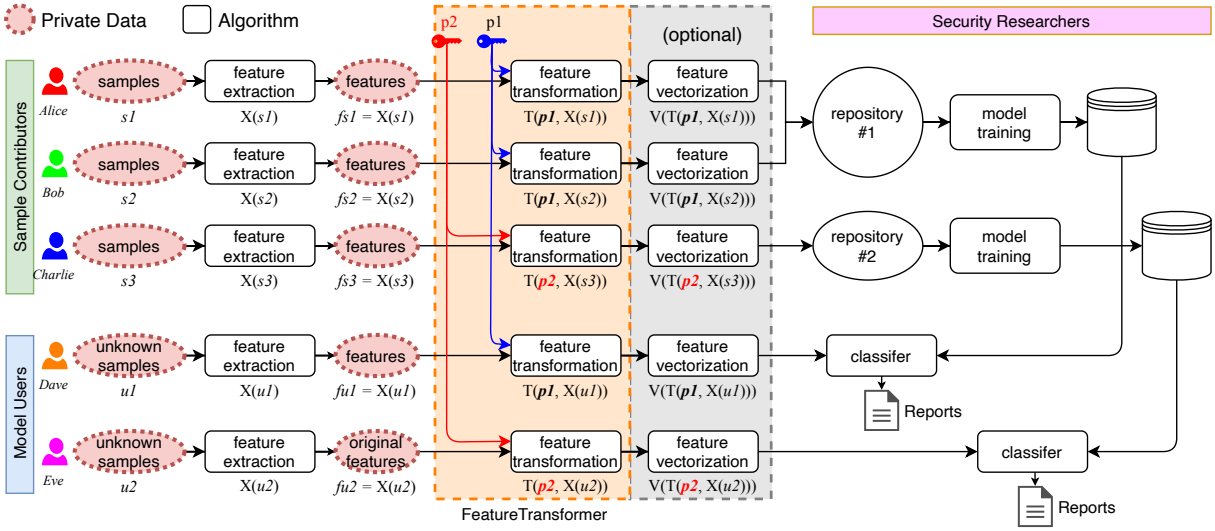


Figure 2: The application scenario of our privacy-preserved dataset sharing model.

researchers, and the model-using phase by model users. A sample contributor collects some raw samples first and then extracts features using feature extraction algorithms in the sample sharing state. Extracted features are passed to FeatureTransformer to transform the features into transformed features. FeatureTransformer is composed of several one-way functions, and therefore, the transformed features are not invertible. Transformed features may need to be vectorized before they can be used for building a classification model. Hence, there is an optional feature vectorization process to ensure that the transformed features meet the requirement of model-building algorithms. The resulted features can finally be stored in the repository. It is worth noting that the parameters of FeatureTransformer must be carefully selected and distributed because it would significantly affect the performance of the system. The impacts of the parameters are discussed in Section 5.

The sample-building phase follows the typical steps in building a machine learning model. Feature samples are retrieved from a repository. After finishing the training process, the resulted models and parameters are stored for future use. In the model-using phase, a user must use the same feature extraction algorithms and feature transformation algorithms and parameters as the sample contributors. Once required features have been extracted and transformed from unknown samples, a classifier can then perform classifications against these transformed features and determine whether given samples are benign or malicious.

4 FEATURE TRANSFORMATION

In this section, we investigate several common malware features first. We then introduce the proposed feature transformation method, FeatureTransformer, to prevent leaking sensitive information. Finally, we discuss the efficiency and the invertibility properties of FeatureTransformer.

Table 1: Common Features for Malware Classification and Detection.

Feature	Sensitive	Datatype	Feature Dimension	In Ember
Bytes Histogram [6, 8, 45]	✗	numerical	256; 257; 256	✓
Entropy [6, 8, 45]	✗	numerical	256; 202; 256	✓
N-gram [5, 6]	✗	numerical	256; 13,000	-
Operation Code Count [6]	✗	numerical	93	-
Register Count [6]	✗	numerical	26	-
Haralick Texture Feature [6]	✗	numerical	52	-
Local Binary Patterns [6]	✗	numerical	108	-
GIST Feature [38]	✗	numerical	320	-
PE Header Info. [5, 8, 45]	▲	numerical	62; 28; 256	✓
Section Info. [5, 6, 8]	▲	numerical	255; 24; 570	✓
Printable String [5, 8]	▲	text-based	104; 26,900	✓
API Sequence [13]	▲	text-based	342	-
API Import [5]	▲	text-based	19,168	-
DLL Import [5, 8, 45]	▲	text-based	1,280; 4,305; 256	✓
Export Function [8]	▲	text-based	128	✓
Raw bytes [41]	✓	text-based	2,000,000	-
Gray Image [51]	✓	text-based	depends on model	-

4.1 Common Features for Malware Research

Table 1 shows the common features used in malware detection studies. We classify listed features into two classes, namely statistical and non-statistical features. Statistical features include byte histogram, entropy, N-gram, opcode count, register count, Haralick texture, and local binary patterns. Generally, statistical features cannot be inverted to samples' original form. Therefore, the risk of leaking sensitive data embeds in malware samples from these features is relatively low.

On the other hand, non-statistical features could be used for information mining, and these features should be protected and transformed. For example, we may extract sensitive information such as the victim's IP, email, or MAC addresses from features such as raw bytes and printable strings. In addition, some features such as *API sequence*, *APP/DLL import*, and *Export information* may not leak sensitive data directly. However, these features still provide

sufficient hints for skillful hackers to learn tricks used in the samples or even identify the corresponding samples on the Internet. For example, if a program calls the functions, RegCreateKeyExA and CryptEncrypt, in ADVAPI32.dll, we can infer that the program tries to modify registers and encrypt data, respectively. Similarly, if malware calls a rarely used function, such as VixDiskMount_RunServer, hackers could directly find the malware sample on the Internet.

4.2 FeatureTransformer

We propose a feature transformation process, FeatureTransformer, to transform features before distributing sample features to the Internet. Generally, features can be classified into two types, text-based and numerical features. We transform text-based features by locality-sensitive hashing (LSH) based algorithms because these algorithms run efficiently and cannot be inverted. Besides, the design principle of LSH ensures that similar inputs have a high probability of transforming to similar results. LSH has many implementations. Among the implementations, we use the hash-based minimization algorithm introduced in MinHash [12] approach to serve as the major algorithm for transforming text-based features in FeatureTransformer.

To differentiate from the *original MinHash*[1] approach, we use the name *minHash* to indicate our hash-based minimization algorithm throughout the paper. The minHash algorithm reads a string D and outputs an h-dimension vector $S = s_1, s_2, ..., s_h$, where h is the number of selected hash functions for minHash algorithm. To obtain S, we first extract a set containing T substrings $d_i, 1 \leq i \leq T$, from D and feed every d_i to all the selected hash functions. The hashed results from hash function $H_k, 1 \leq k \leq h$, for substring d_i is $H_k(d_i)$. Then, the minimal result s_k of all $H_k(d_i), 1 \leq i \leq T$ with respect to the k-th hash function can be obtained. The process can be formulated with

$$s_k = \min_{d_i \in D} H_k(d_i), 1 \leq k \leq h. \quad (1)$$

For numerical features, we employ two transformation methods. One is to handle numerical features as strings and process the *numerical string* by LSH-based algorithms. The other is a matrix-based transformation algorithm inspired by the minHash algorithm. The matrix-based transformation algorithm includes a matrix multiplication, a minimization process, and a modulo operation. A transformed feature is in the same form as the original feature. Thus, for example, a transformed text-based feature has the same length as the original feature. Similarly, a transformed numerical feature also has the same form (a single number or a matrix of the same dimension) as the original feature. The details of how to transform text-based and numerical features are discussed as follows.

Text-based Features. We use the minHash algorithm as the major component to handle text-based features. Figure 3 depicts an example of the text-based feature transformation using the minHash algorithm. We segment a string feature into non-overlapping substrings of length W as the inputs to the minHash algorithm. For each substring D, we then obtain d_i from D by using N-gram [26, 49]

[1]The original MinHash approach calculates the similarity of hashed-dataset via set similarity measurement functions like Jaccard Similarity. Right here, our minhash algorithm only converts the output hashed result into a string.

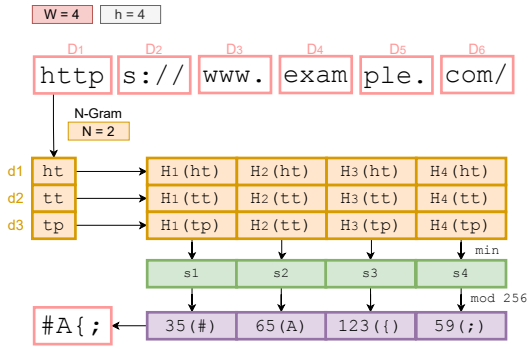


Figure 3: An example of text-based feature transformation using minHash.

approach. After feeding d_i to hash functions of the minHash algorithm, we can obtain the output result S. We perform modulo operations against each element in S. Finally, we concatenate the results in S as the final output string, which is the transformed results. Given that hash outputs are modules of 256, we set $h = W$ to ensure the length of a transformed string would be equivalent to the length of the input D. For example, in Figure 3, the string "`http`" is transformed to the corresponding result "`#A{;`". The characteristic of the minHash algorithm ensures that similar inputs would have a high probability of having similar results. Although the transformed results may lose some information, it ensures that similar strings can still be placed closely in the transformed domain. Moreover, since the transformed results cannot be inverted, it successfully prevents sensitive information from being leaked.

Numerical Features. We propose two transformation methods to handle the numerical features, (1) transforming by the minHash algorithm and (2) transforming by numerical matrices. For the first method, we transform the numerical data to string data. For example, number 1 is transformed to string "$\backslash x01$". We prepend null bytes before the transformed string if the string is shorter than W bytes. Then, the transformed numerical features can be processed by the minHash algorithm. Once the transformed results are obtained, we then transform the results back to integers. For the matrix-based transformation method, we perform the following steps to transform numerical features F in the form of a $1 \times V$ matrix. First, we prepare a set M containing R two-dimensional $V \times V$ matrices, $M = \{m_1, m_2, ..., m_R\}$ and assign random values to elements of each of the R matrices. We then perform R matrix multiplications to multiply the numerical feature matrix F against each of the R matrices and obtain the results in a set Z, where $Z = \{z_1, z_2, ..., z_R\} = \{F \times m_1, F \times m_2, ..., F \times m_R\}$. Finally, we output a transformed feature F' by selecting a minimal element from each column of matrices in Z. That is,

$$F' = [\min_{i=1...R} z_{i_{1,1}}, \min_{i=1...R} z_{i_{1,2}}, ..., \min_{i=1...R} z_{i_{1,R}}]. \quad (2)$$

An optional modulo operation can be applied to each element in F' to produce the final transformed feature vector.

Table 2: The minHash Transformation Throughput (bytes/sec) for Different W and N Combinations.

N-gram	W = 1	W = 2	W = 4	W = 6	W = 8
N = 1	113,904	139,840	131,410	111,275	92,278
N = 2	-	188,499	158,568	128,646	102,934
N = 3	-	-	205,370	150,101	118,656
N = 4	-	-	304,874	181,110	138,961
N = 5	-	-	-	247,361	166,284
N = 6	-	-	-	387,023	207,178
N = 7	-	-	-	-	277,045
N = 8	-	-	-	-	441,843

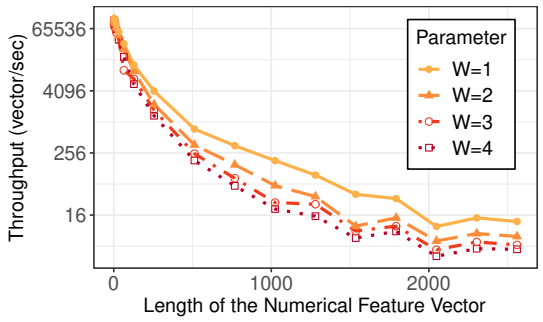


Figure 4: The transformation throughput (vector/sec) of numerical features for different W.

4.3 Properties of FeatureTransformer

We discuss two properties of FeatureTransformer and how involved parameters affect these properties.

Efficiency. Table 2 shows the transformation throughput under different combinations of W and N values. All the measurements are conducted on a server machine with an Intel® Xeon® Gold 5218 CPU running at 2.30GHz. Given a selected W value, we observe that the transformation throughput and N have a positive correlation. The value N decides the number of elements d_i in each D and would have a linear decrease in the number of required computations. However, the throughputs presented in Table 2 may not grow linearly with N because we employ an LRU cache mechanism to eliminate redundant hash operations on the same inputs. In contrast, given a selected N value, the total number of string D derived from a string decreases when W increases. However, the number of elements d_i in each D increases accordingly. Since our text-based transformation approach is designed to produce exactly the same output of length equivalent to the input string, the number of hash functions h increases with W. Therefore, it also increases the required hash computations. We note here that (1) a smaller N could lose character sequence information from the original input data. For example, when N is equal to one, hashing four characters independently from string "http" and "ptth" would get the same results; (2) when N is equal to W, the input D is directly fed to the hash functions. The measured throughput for all W and N combinations achieves 200K bytes per second, which shows the efficiency of FeatureTransformer for handling text-based features. The transformation throughput is bounded by the length of the numerical feature and the matrix multiplication for numerical features. We conduct a simple experiment to investigate the throughput. Specifically, we randomly generate 10,000 different numerical feature vectors F with a given length. Then, we multiply F with the corresponding generated R two-dimension matrices M and record the total running time to estimate the throughput. Figure 4 shows the experiment result. We observe that the throughput has a negative correlation to the value R and the length of the numerical features. The result is intuition because a larger R or vector length implies that more multiplication operations are involved.

Invertibility. FeatureTransformer employs multiple hash functions, multiple matrix multiplications, and minimization processes to ensure that input D cannot be inverted. We investigate the invertibility of FeatureTransformer from the perspective of parameter selection, e.g., W and N. In general, increasing hash collisions would also increase the difficulties of inverting hashed outputs. For example, suppose hashing T inputs produces another T distinct outputs. In this case, it is evident that the outputs could be easily mapped back to the original inputs if a sufficient number of inputs and outputs are collected. Therefore, ensuring a certain amount of collision would be the key to ensuring the invertibility property of FeatureTransformer. We illustrate the invertibility property using a simple case of setting $W = 3$ and altering the value of N. Assume all hash functions do not produce collisions. When N is set to one, the number of possible transformed output strings from the minHash algorithm is only 2,796,416, which is calculated by the formula $C_3^{256} + C_2^{256} + C_1^{256}$ (given the modulus parameter of 256). It is because feeding any permutation of three selected characters into the minHash algorithm would produce the same result. Therefore, we can simply count the possible number of mathematical combinations for strings of length three. While the possible permutations of input strings are 16,777,216 (2^{24}), the high collision rate makes it challenging to invert output strings back to their original forms.

The minimization and modulus operation would further increase the collision probabilities and hence increase the invertibility of transformed features. Table 3 shows the simulated statistics for minHash collisions under different settings of W and N parameters. The size of the input space is the number of possible inputs, which can be obtained by 256^W. The size of output-space is the number of distinct minHash outputs from all possible inputs. The hash functions selected for the minHash algorithm are MD5, SHA256, and SHA384. For each (W, N) combination, we simulate ten rounds and choose a randomly generated salt of length 6 in each round to ensure the hash functions would not produce the same results in different rounds. The results show that a lower N value would increase more collisions, and there are only at most 36% inputs without collisions even if N is equal to W. However, too many collisions may lead to the loss of crucial information used to differentiate inputs. Therefore, our experiments use $N = 2$ if not mentioned otherwise to balance the invertibility and the loss of crucial information based on the observations.

One benefit of the invertible property used in FeatureTransformer is that it also provides the access control feature. Given

Table 3: Simulated Statistics for minHash Collisions.

(W,N)	output-space/input-space	no-collision
(1,1)	60%	35%
(2,1)	16%	0%
(2,2)	63%	36%
(3,1)	3%	0.00009%
(3,2)	43%	24%
(3,3)	63%	36%

that features transformed with different parameters, e.g., randomly generated hash salts and matrix elements, would be pretty diverse. These randomly generated parameters can be considered unique keys to control dataset accessibility. Users without the key can still use the transformed features to train models, but they cannot add new samples to the shared dataset. They also cannot use trained models to detect and classify samples out of the shared dataset. Readers can imagine that the original sample features in vector space F are non-linearly transformed into another vector space T using parameters in set P. As a result, a model M built with features in space T can only be used for classifying sample features in space T. The invertibility property ensures that it is infeasible to obtain features in F from features in space T. Therefore, the only way to use model M for classifying unknown samples is to transform sample features to the same space T, which requires the knowledge of P. The effectiveness of access control feature is further discussed in Section 5.5.

5 EVALUATION

This section validates our proposed FeatureTransformer by answering the research questions (RQ) mentioned above. We evaluate FeatureTransformer by using a binary malware classification problem and answer RQ1 to RQ5. For RQ6, we evaluate FeatureTransformer by using a generic binary image classification problem. The involved datasets are summarized as follows:

- We use the EMBER 2018 dataset [8] to experiment with the malware classification problem. The EMBER dataset contains 1.1 million samples split into a training set (900K) and a testing set (200K). The samples in the training set are labeled as malware (300K), benign (300K), and unlabeled (300K). In contrast, the samples in the testing set are only labeled as malware (100K) and benign (100K).
- We use the Kaggle dataset, Dogs vs. Cats [2], to experiment with the generic image classification problem. The dataset contains 25,000 images, of which 12,500 and 12,500 images are dogs and cats, respectively.

5.1 Experiment Setup

For the malware classification problem, FeatureTransformer transforms extracted features (EFs) available in the EMBER dataset. There are nine features in the dataset. Two of them are text-based features (*Imports Information (II)* and *Exports Information (EI)*). The rest seven are numerical features (*Header File Information (HFI)*, *General File Information (GFI)*, *String in the Raw Data (SRD)*, *Byte Histogram (BH)*, *Byte Entropy Histogram (BEH)*, *Data Directories (DD)*), and *Section Information (SI)*). To train the classifiers, we randomly sample 100,000 malware and 100,000 benign samples from the EMBER dataset. We then split samples into a training set and a validation set randomly in a ratio of 80:20, respectively. To investigate the performance of on handling both text-based and numerical features, we consider three different feature combinations, which include (i) transforming only text-based features and training a model with only transformed features (F2T2); (ii) transforming only text-based features but training a model with both transformed text-based features and original numerical features (F9T2); and (iii) transforming all the features and training a module with all the transformed features (F9T9). For each feature combination, we employ several commonly used binary classifiers include Logistic Regression (LR), Linear Support Vector Classifier (LSVC), AdaBoost (AB), Decision Tree (DT), Random Forest (RF), Gradient Boosting (GB), and LightGBM (LGBM).

For the image classification problem, we also split samples into a training and a validation set in a ratio of 80:20, respectively. A CNN model, ResNet50 [22], is employed to build the binary image classifier. We follow the same settings used in the RestNet50 paper. Note that FeatureTransformer performs transformation against raw pixels directly because the ResNet50 model uses raw image pixels as its input. All the experiments relevant to a built model are evaluated with 5-fold cross-validation to ensure the model has consistent experiment results.

[2]https://www.kaggle.com/c/dogs-vs-cats/data

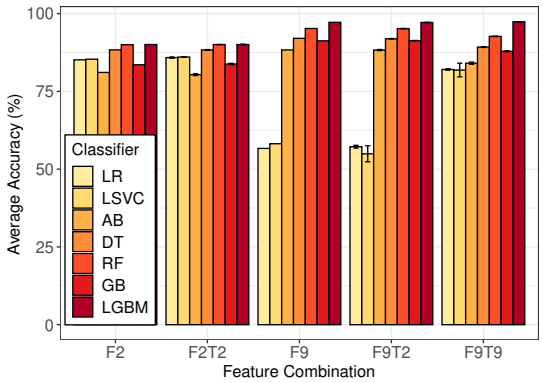


Figure 5: The average accuracy of different feature combinations.

5.2 RQ1: Does FeatureTransformer Really Work?

We first show the performance of models built based on features transformed by FeatureTransformer. Figure 5 shows the average detection accuracy of different feature combinations. Due to space limitations, we only show detection accuracy here. The results for other performance metrics are similar to detection accuracy. Readers may refer to the discussions in Section 5.3 for the results of different performance metrics. Overall, the performance numbers of a classifier are similar among different feature combinations. Due to negligible performance differences for the two text-based

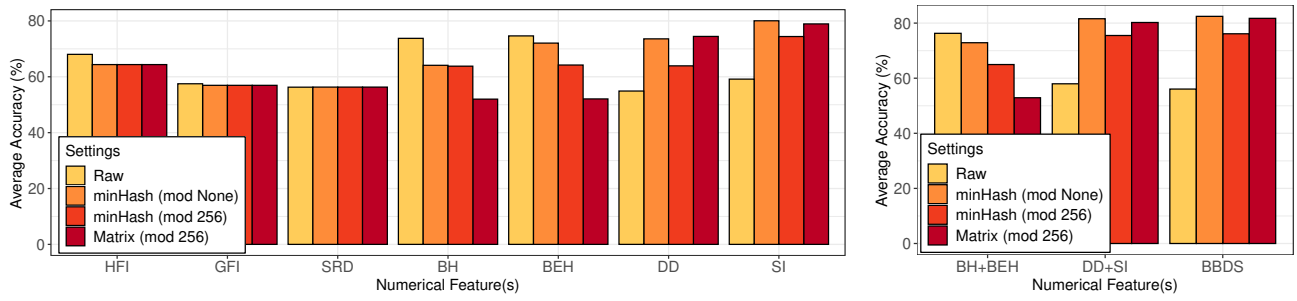


Figure 6: The average accuracy of (a) numerical features and (b) the combinations of numerical features on different settings.

features (F2 vs. F2T2), another question is raised: what could be the performance impacts of mixing transformed features with untransformed features? Therefore, we further conduct F9, F9T2, and F9T9 experiments, which mixed the two transformed/untransformed text-based features with seven numerical features. The outcome also shows that the performance remains consistent no matter the text-based features are transformed.

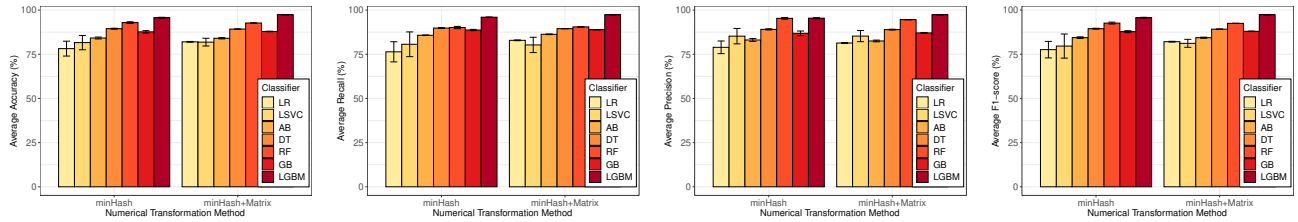
Readers may notice that the two linear classifiers, LR and LSVC, get lower accuracy than other classifiers on the F9T2 combination. However, the performance aligns with those working with the original numerical features (see bars of group F9). The impact of different parameters settings (W and N) are later discussed in Section 5.4. Interestingly, the detection accuracy significantly increases on the F9T9 combination, indicating that feature transformation would benefit classifiers. We suspect that the lower detection accuracy on original numerical features is because linear classifiers cannot effectively classify involved features. As a result, it leads to worse performance on LR and LSVC classifiers. Once numerical features are transformed into new feature spaces, such as the F9T9 combination, the samples may be split linearly.

To validate our speculation, we perform malware classification only with the untransformed or transformed numerical features using LR and LSVC. Four transformation settings are tested in this experiment, (i) no transformation (Raw), (ii) transformed by minHash without a modulus, (iii) transformed by minHash with a modulus of 256, and (iv) transformed by Matrix method with a modulus of 256. While the results for the LR and LSVC classifiers are similar, we only plot results for LR due to space limitations. Figure 6(a) shows the average accuracy of the classified results for each numerical feature. While we expect FeatureTransformer would get a similar or slightly lower performance than those trained with original features, we notice that the results are better than our expectations. We observe that transformed numerical features can have (i) no impact, (ii) positive impacts, and (iii) negative impacts on the detection accuracy. Features such as HFI, GFI, and SRD have no impact after feature transformation and get similar performance for both raw features and transformed features. Features such as DD and SI get much better performance after being transformed. The only two transformed features that get lower performance are BH and BEH. Both BH and BEH are histogram-based features, meaning that numbers in the vectors are highly coupled with each other. When applying hash and modulus operations to histograms, it breaks the relationships between numbers in the vectors. We further train classifiers based on the combinations of these numeric features, including (i) BH+BEH, (ii) DD+SI, and (iii) BH+BEH+DD+SI (BBDS). Figure 6(b) shows the result. While BH+BEH and DD+SI show consistent performance to what we observed for histogram and non-histogram features, respectively, we observe that BBDS get much better performance. The result aligns with what we reported in F9T9 in Figure 5.

5.3 RQ2: Performance of Text-based and Numerical Feature Transformation

As we introduced in Section 4.2, text-based features are transformed by LSH-based algorithms, and numerical features can be transformed by either LSH-based algorithms or matrix operations. In this section, we conduct experiments to show the performance of employing different transformation methods. The LSH-based algorithm used here is the minHash algorithm. The algorithm parameters are set to $W = 4$ and $N = 2$.

Figure 7 shows the experimental results on the accuracy, recall, precision, and F1-score. The error bar on top of each bar in the figures indicates the range of one standard deviation. The group of bars labeled **minHash** indicates that all features are transformed by the minHash algorithm, whether text-based or numerical. The group of bars labeled **minHash+Matrix** indicates that text-based and numerical features are transformed using the minHash algorithm and matrix operations, respectively. Although the performance trends on different feature combinations look similar, our results show that working with matrix operations performs slightly better than those only transformed by the minHash algorithms. The minHash+Matrix setting outperforms the minHash-only setting by 0.77%, 1.18%, 0.45%, and 1.07% on average for accuracy, recall, precision, and F1-score, respectively. The discussion of the performance of the individual classifier is out of the scope of this paper. However, we can observe that a linear classifier, i.e., LR and LSVC, generally performs worse than tree-based classifiers. Overall, LightGBM achieves the best performance in the experiments, aligning with other research papers using the EMBER dataset [8, 21]. In summary, we conclude that both text-based and numerical feature



(a) Accuracy (b) Recall (c) Precision (d) F1-score

Figure 7: The evaluation results for the text-based and numerical feature transformations on the F9T9 feature combination.

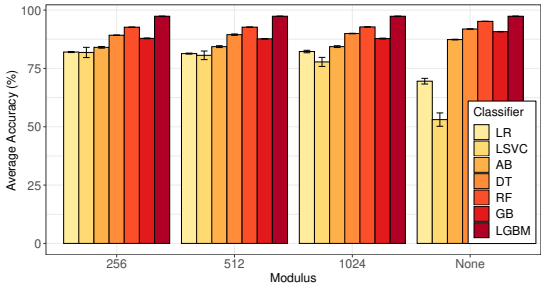
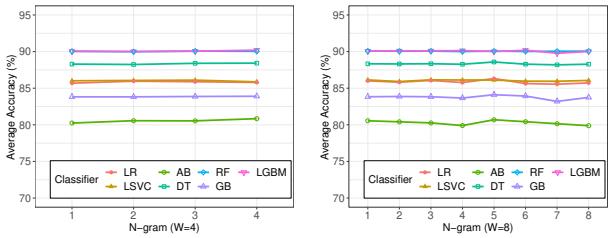


Figure 8: The average accuracy under different modulus parameters for numerical features on the F9T9 feature combination.



(a) (b)

Figure 9: The average accuracy for different minHash parameters, W and N, on the F2T2 feature combination.

transformations work well. If detection performance is pretty demanding, it would be better to work with the *minHash+Matrix* approach.

5.4 RQ3: Impacts of Algorithm Parameters

In this section, we investigate the impact of different parameter values of FeatureTransformer. We compare the performance by varying the algorithm parameters in FeatureTransformer, including (i) the modulus for numerical features, (ii) the length W of the input string for the minHash algorithm, and (iii) the parameter N for the N-gram algorithm to generate substrings from an input string.

Modulus for numerical features. For text-based features, we always use a modulus of 256 to ensure the output string has exactly the same length as the input string; for numerical features, we attempt to seek the best arithmetic modulus. We explore the impact of different modulus parameters, including 256, 512, 1024, and no modulus (None). Figure 8 shows the result of the experiment. We observe that the differences among the different modulus parameters are not significant except for the LR and the LSVC classifier in setting "None." Nevertheless, when the numerical features are transformed with an additional modulo operation, the detection accuracy generally increases, and the detection performance is indifferent to modulus values. Therefore, we set the modulus parameter to transform numerical features to 256 to report and analyze the results in the following sections.

Parameters for minHash. To investigate the algorithm parameters of minHash, we jointly consider the different combinations of W and N parameters. Because the minHash algorithm is mainly designed for text-based features, we experiment on the F2T2 feature combination to avoid performance impacts caused by numerical features. Figure 9 shows the experiment results for different N values given $W = 4$ or $W = 8$. Although there is a noticeable performance difference among classifiers, varying W and N does not affect a classifier's performance. Among different algorithm parameters, the detection accuracy is indifferent to the values. Specifically, in the $W = 4$ and $W = 8$ experiments, the performance differences are less than 0.6% and 1%, respectively, for a measured classifier on different N values. Although we have already zoomed in on the Y-axis to highlight the performance difference, a few lines still overlap with others. For example, the LR/LSVC classifiers in the W=4 experiment and the RF/LGBM classifiers in both the W=4 and W=8 experiments. Based on the experiment results, we conclude that the algorithm parameters of minHash have a limited impact on the detection performance, which shows the robustness of FeatureTransformer.

5.5 RQ4: Access Control

As we introduced in Section 3.2, our proposed FeatureTransformer provides two different access privileges: complete and limited. We achieve this goal by using the transformation parameters as the key to granting access privileges. A user has the complete access privilege if and only if the user knows all the parameters to perform the transformation. In contrast, a user only has a limited access privilege if the user does not know the parameters. Note that a user with a limited access privilege can still use the transformed feature dataset to train and evaluate the models. However, she/he cannot add new samples into the dataset nor use a built model to detect untransformed samples. We conduct experiments to verify

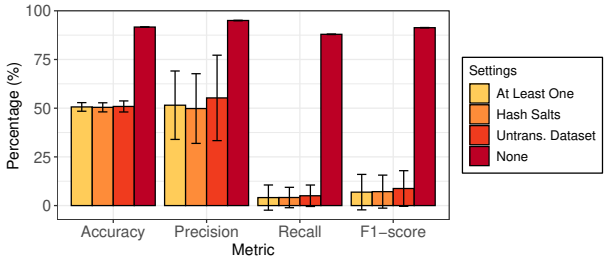


Figure 10: The detection performance for the four experiment settings: (i) at least one parameter is unknown, (ii) the hash salts for minHash algorithms are unknown, (iii) the untransformed dataset, and (iv) no parameter is unknown.

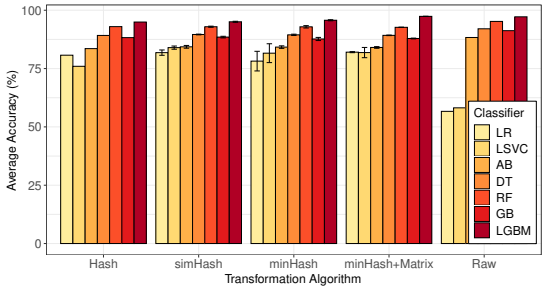


Figure 11: The average accuracy under different transformation approaches on F9T9 feature combination.

our design. The involved parameters include W, N, transformation methods, and hash functions. In this section, we compare the performance of four settings, (i) at least one parameter is unknown, (ii) the hash salts for minHash algorithms are unknown, (iii) the untransformed dataset, and (iv) no parameter is unknown. The experiments are conducted as follows. Note that while settings (i), (ii), and (iv) are relevant to parameter selections, we add an additional experimental setting (iii) to show that a transformed model can fully eliminate possibly information leakage from the original (untransformed) dataset. First, we prepare 50,000 samples from the EMBER dataset and use 80% and 20% of the samples to train and validate a model. Second, we select a set of parameters S and transform the features of the 80% samples and use the transformed features to train a model. Third, we substitute unknown parameters in S and transform the features of the rest 20% of samples. Transformed features are then classified by using the model built in the second step. Note that if a parameter is unknown depending on the experiment settings, we use a randomly generated value to substitute the unknown one when transforming the features. We use LightGBM as the classifier to conduct the experiments because it has the best performance among all the classifiers. We employ F2T2 feature combination, which considers only text-based features. We run 5-fold cross-validation for all the experiment settings.

Figure 10 shows the experiment results. We can observe that if any one of the parameters is unknown, the performance would significantly decrease. The result is intuition because the features transformed by different parameters are placed in a different space and have a different distribution. Therefore, the models trained from a given set of transformed features cannot effectively classify samples transformed with different parameters. We can also observe that the performance are similar in experiment settings (i), (ii), and (iii), which means that any minor change in the parameters would lead to massive performance degradation. Based on the experiment results, we conclude that using transformation parameters as the key is effective for access control.

5.6 RQ5: Impacts of Transformation Algorithms

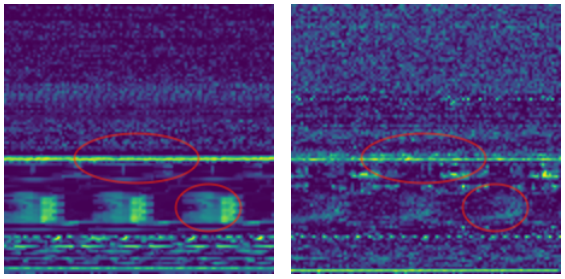
We compare different transformation methods for both text-based and numerical features to test the flexibility of FeatureTransformer. We replace the default text feature transformation algorithm (minHash) of FeatureTransformer with one of the follows:

- **Hash**. We directly hash the raw bytes of length W from a malware sample with a SHA3-512 hash function and extract the last W bytes of the hashed result as the classifier inputs.
- **simHash**. We replace the minHash algorithm with the SimHash algorithm [14], which is also an LSH-based algorithm.
- **minHash**. We use the minHash algorithm to transform both text-based and numerical features.
- **minHash+Matrix**. We use the minHash algorithm and the matrix transformation approach to transform the text-based and numerical features, respectively.
- **Raw**. We use the untransformed features to train the classifier.

The experiment is conducted with the F9T9 feature combination to investigate the transformation algorithms' performance impacts thoroughly. Figure 11 shows the experiment result. The no transformation setting (Raw) achieves the best detection accuracy, except for the LR and LSVC algorithms. The result is not surprising because the raw samples contain information without any transformation loss. The lower detection accuracy for LR/LSVC classifiers is because of the capability limitation of linear classifiers, as we discussed in Section 5.2. Among the transformation algorithms, the minHash+Matrix transformation achieves the best detection accuracy for all classifiers. The performance is pretty close to those performed on untransformed features. Specifically, the difference in detection accuracy between minHash+Matrix and Raw are -4.27%, -2.81%, -2.53%, -3.36%, and 0.19% for classifier AB, DT, RF, GB, and LGBM, respectively. Nevertheless, the performance difference among the algorithms is insignificant. This experiment shows the flexibility, robustness, and stability of FeatureTransformer. Users can select different transformation algorithms to meet their demands.

5.7 RQ6: Adopting FeatureTransformer on CNN-based Malware Detection Classifiers

We further evaluate the performance of FeatureTransformer with neural-network-based models. Inspired by Nataraj et al. [38], we consider an extreme case that converts an entire binary file as an image and feeds the image to an image classification model. However, instead of feeding the original image, we transform image pixels using the minHash algorithm, feed transformed images into the network, and measure and compare the classification performance.

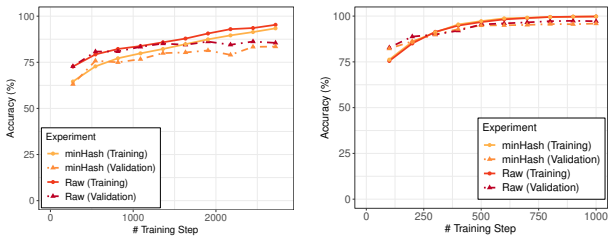


(a) Raw (b) minHash

Figure 12: A sample Grad-CAM results of the fifth hidden convolutional layer for different inputs (a) untransformed (Raw) and (b) transformed by the minHash algorithm.

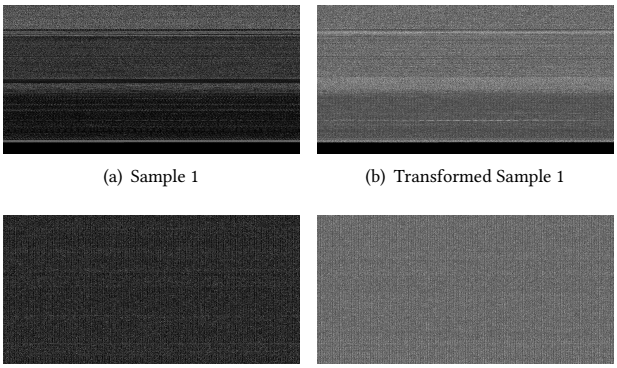
In the experiments, we first convert the first 2MB of each malware sample to a two-dimension (2D) image based on the approach proposed by Nataraj et al. [38]. We then use a modern 2D-CNN image classification network VGG-16 [48] to serve as the classifier to classify untransformed and transformed images. To further explore what the classifier learns, we adopt Grad-CAM [46] to visualize the responses of the last hidden convolutional layer right before the output layer. Figure 12 shows a sample Grad-CAM results of the *fifth* hidden convolutional layer for untransformed and transformed inputs. We can observe that the highlighted parts in the figures are spatially similar to each other. Figure 13(a) shows the classification accuracy for untransformed and transformed features. It shows that working with untransformed features (Raw) achieves 86.23% (95.31%) validation (training) accuracy. In contrast, working with transformed features (minHash) achieves 79.97% (93.40%) validation (training) accuracy. We first observe that the training and detection accuracy is similar between the untransformed and transformed features. Second, the obtained models look overfitting to the training data when the training steps exceed 1,500. We can see that the validation accuracy stops increasing and maintains accuracy at about 86% and 80%, respectively, for the Raw and minHash settings. Third, we notice that the validation samples get a lower detection performance. We suspect that the lower validation accuracy could be mainly caused by converting a binary file to a 2D image, in addition to the possible overfitting. Two-dimension convolution operations integrate spatial information from nearby pixels assuming that pixels within a small region should be highly relevant. However, the assumption may not apply to images converted from malware sample files. If we incorrectly perform convolution against irrelevant pixel data, it could be harmful to the detection accuracy.

To validate our speculation, we convert samples to a 1D feature based on the approach proposed by Raff et al. [41]. Specifically, we convert the first 2MB of file content byte-by-byte to a vector of 2M dimensions. We then feed the 1D vector to a 1D-CNN classifier. Figure 14 shows two samples and their corresponding transformed results. We can observe that the patterns are similar between an original sample and its transformed form. The learning curve for the 1D-CNN classifier is depicted in Figure 13(b). We can observe that the 1D-CNN classifier gets similar performance for both untransformed and transformed features. The performance of the



(a) 2D-CNN (b) 1D-CNN

Figure 13: The learning curves for the training and validation sets in the (a) 2D-CNN and (b) 1D-CNN experiments.



(a) Sample 1 (b) Transformed Sample 1

(c) Sample 2 (d) Transformed Sample 2

Figure 14: We transform the first 2M bytes of a sample to a 1D vector by using the minHash algorithm. The vectors are rendered as 2D images for readability.

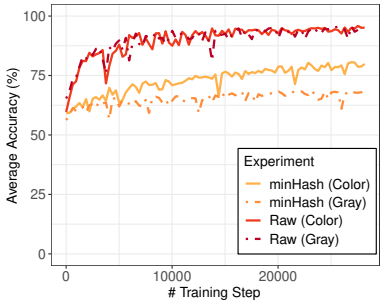
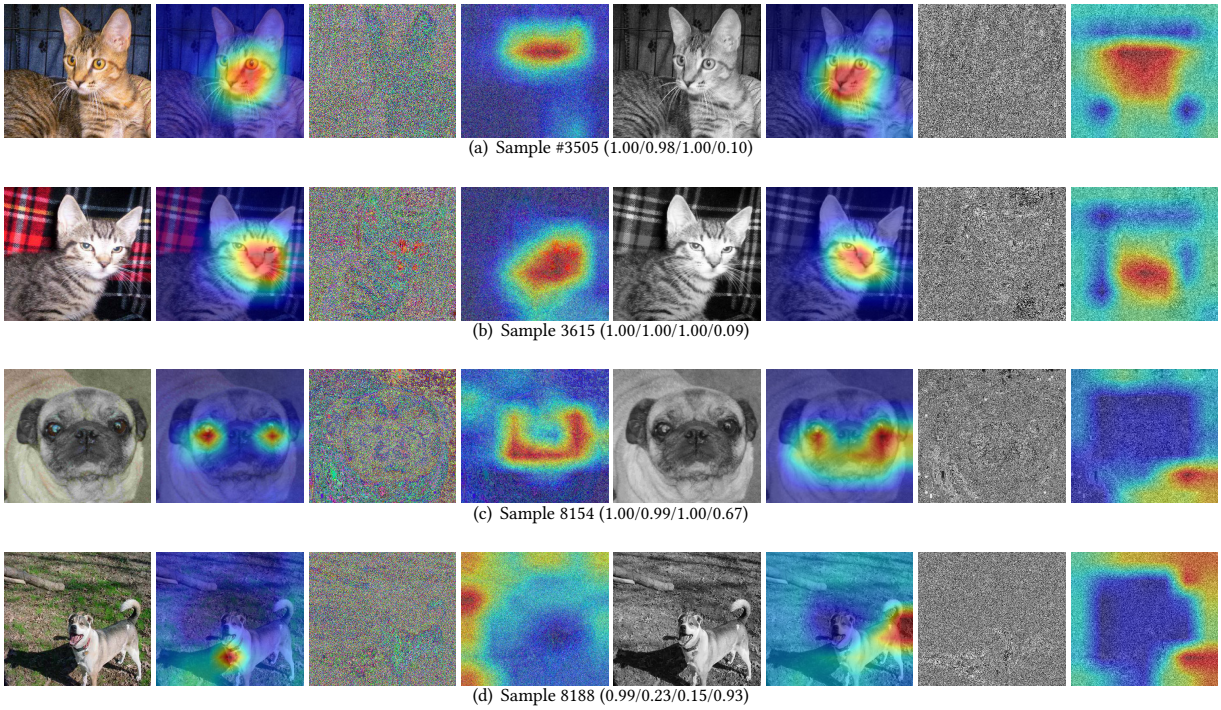


Figure 15: The learning curves for the training and validation sets used in the generic image classification problem.

validation dataset is also close to that of the training dataset. In summary, we conclude that FeatureTransformer generally works for neural-network-based classifiers. The neural network design may affect the performance of the classifier, but the performance trends remain similar for both untransformed and transformed features.



(a) Sample #3505 (1.00/0.98/1.00/0.10)

(b) Sample 3615 (1.00/1.00/1.00/0.09)

(c) Sample 8154 (1.00/0.99/1.00/0.67)

(d) Sample 8188 (0.99/0.23/0.15/0.93)

Figure 16: Selected samples and their corresponding Grad-CAM results for classifiers built based on the different datasets. The values in the parentheses are the confidence values reported from the corresponding classifiers (RGB image/Transformed RGB image/Gray-scale image/Transformed gray-scale image).

5.8 RQ7: FeatureTransformer for a Generic Image Classification Problem

We also apply FeatureTransformer to a generic image classification problem to test the generality of our proposed approach. This experiment uses the Dogs and Cats classification dataset from Kaggle [27] by FeatureTransformer. Animal images are transformed by FeatureTransformer and then classified using the classifier proposed by ResNet50 [22] . We design two different scenarios to conduct the experiments. One is conducted based on the untransformed images, and the other is conducted based on the transformed images. In the first scenario, we use RGB values of a pixel as the input and transform the input using the minHash algorithm with W=3 and N=1. The setting is under the assumption that RGB colors are i.i.d. In the second scenario, we convert RGB images to gray-scale images and then perform the transformation with W=1 and N=1. Figure 15 shows the learning curve of the experiments. In the experiments, the classifiers trained by the RGB or gray-scale images both achieve above 95% of detection accuracy. For the classifiers trained by the transformed RGB and gray-scale images, the detection accuracy is about 82% and 70%, respectively.

We further look into some examples from the detection results. For each experiment, we use Grad-CAM to highlight the hot areas learned by the classifier. Figure 16 shows samples of the raw images, transformed images, and their corresponding Grad-CAM results on the fourth hidden layer (the last hidden layer right before the output layer). There are four rows in the figure. Each row contains four sample images from the two experimental scenarios, including the original RGB image, the transformed RGB image, the gray-scale image, and the transformed gray-scale image. There is also a corresponding Grad-CAM result right next to each sample image. We can observe that the transformed images preserve some outlines in the raw images. Although the outlines may be unrecognizable to humans, the classifier can still handle transformed images. Note that the values enclosed in the parentheses of sub-figure captions are the confidence values reported from the classifiers built based on the corresponding datasets. The four numbers from left to right are values for the RGB images, the transformed RGB images, the gray-scale images, and the transformed gray-scale images.

In Figure 16, the samples rendered in the first three rows are the correctly classified cases, and the samples in the last row (except the original RGB sample) are the failure cases. Note that the bottom right transformed gray image can be correctly classified, but the reported hot area looks irrelevant to the identified dog. Since transformed images would drop critical features used to classify the images, it is expected that the detection accuracy would be lower than working with the original images. However, our experiments show that although transformed images could be broken, classifiers can still correctly classify most images. In the correctly classified cases, we can see that the identified hot areas by the classifiers are similar given different training and validation datasets. We also placed a few incorrectly classified cases in Figure 16. The classifiers select incorrect areas (the grass fields in the corners) to decide in

the transformed images. Similar failure also happens on the gray-scale image. Although the confidence value for the transformed gray-scale image is pretty high, we believe that it was just lucky based on hot areas reported from Grad-CAM.

Overall, although the transformation process inevitably loses some information of the original images and drops the detection accuracy, the detection accuracy is still significantly better than random guesses for a binary classification problem, namely, 50%. Meaningful patterns are preserved after feature transformation and can still be recognized by the classifier.

6 CONCLUSION

A representative dataset is essential for research works using machine learning approaches. It would help researchers to evaluate proposed solutions fairly and conduct reproducible experiments. However, the possible leakage of private information from security datasets impedes the sharing of samples. We proposed a privacy-preserved dataset sharing model to eliminate the possible leakage of sensitive information to increase sharing. Our proposed approach, FeatureTransformer, handles both text-based and numerical features with a specially designed transformation algorithm based on locality-sensitive hashing algorithms and matrix-based operations. We conduct a series of experiments to validate the effectiveness of FeatureTransformer. The experiment results show that FeatureTransformer is a practical solution without significant performance degradation. We believe that our privacy-preserved dataset sharing model sheds light on building a Utopia of security dataset sharing. By stimulating dataset sharing, more reproducible machine-learning-based security works can be fairly evaluated and further deployed for solving real-world problems.

ACKNOWLEDGMENT

This study is supported in part by the Ministry of Science and Technology under grants number MOST 110-2628-E-A49-011, and MOST 110-2218-E-A49-011-MBK, and in part by the Center for Open Intelligent Connectivity from The Featured Areas Research Center Program within the framework of the Higher Education Sprout Project by the Ministry of Education (MoE) in Taiwan. We would also like to thank the anonymous reviewers for their constructive and insightful comments.

REFERENCES

[1] [n.d.]. VirusTotal. https://www.virustotal.com/

[2] 2018. A free resource for researchers and practitioners to find and follow the latest state-of-the-art ML papers and code. Online. https://paperswithcode.com/.

[3] 2019. APT Malware Dataset. Online. https://github.com/cyber-research/APTMalware.

[4] Martin Abadi, Andy Chu, Ian Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. 2016. Deep Learning with Differential Privacy. *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (Oct 2016). https://doi.org/10.1145/2976749.2978318

[5] Hojjat Aghakhani, Fabio Gritti, Francesco Mecca, Martina Lindorfer, Stefano Ortolani, Davide Balzarotti, Giovanni Vigna, and Christopher Kruegel. [n.d.]. When Malware is Packin' Heat; Limits of Machine Learning Classifiers Based on Static Analysis Features. *Network and Distributed Systems Security (NDSS) Symposium 2020* ([n. d.]). https://doi.org/10.14722/ndss.2020.24310

[6] Mansour Ahmadi, Dmitry Ulyanov, Stanislav Semenov, Mikhail Trofimov, and Giorgio Giacinto. 2016. Novel Feature Extraction, Selection and Fusion for Effective Malware Family Classification. In *Proceedings of the Sixth ACM on Conference on Data and Application Security and Privacy* (New Orleans, Louisiana, USA) *(CODASPY '16)*. ACM, New York, NY, USA, 183–194. https://doi.org/10.1145/2857705.2857713

[7] A. S. Ajeena Beegom and Gayatri Ashok. 2020. Malware Detection in Android Applications Using Integrated Static Features. In *Security in Computing and Communications*, Sabu M. Thampi, Gregorio Martinez Perez, Ryan Ko, and Danda B. Rawat (Eds.). Springer Singapore, Singapore, 1–10.

[8] H. S. Anderson and P. Roth. 2018. EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models. *ArXiv e-prints* (April 2018). arXiv:1804.04637 [cs.CR]

[9] Zahra Bazrafshan, Hashem Hashemi, Seyed Mehdi Hazrati Fard, and Ali Hamzeh. 2013. A survey on heuristic malware detection techniques. In *The 5th Conference on Information and Knowledge Technology*. 113–120. https://doi.org/10.1109/IKT.2013.6620049

[10] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konecny, Stefano Mazzocchi, H. Brendan McMahan, Timon Van Overveldt, David Petrou, Daniel Ramage, and Jason Roselander. 2019. Towards Federated Learning at Scale: System Design. http://arxiv.org/abs/1902.01046 cite arxiv:1902.01046.

[11] Michael Brengel and Christian Rossow. 2021. YARIX: Scalable YARA-based Malware Intelligence. In *USENIX Security Symposium*.

[12] A. Z. Broder. 1997. On the resemblance and containment of documents. In *Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No.97TB100171)*. 21–29. https://doi.org/10.1109/SEQUEN.1997.666900

[13] Ferhat Ozgur Catak, Ahmet Faruk Yazı, Ogerta Elezaj, and Javed Ahmed. 2020. Deep learning based Sequential model for malware analysis using Windows exe API Calls. *PeerJ Computer Science* 6 (July 2020), e285. https://doi.org/10.7717/peerj-cs.285

[14] Moses S. Charikar. 2002. Similarity Estimation Techniques from Rounding Algorithms. In *Proceedings of the Thiry-Fourth Annual ACM Symposium on Theory of Computing* (Montreal, Quebec, Canada) *(STOC '02)*. Association for Computing Machinery, New York, NY, USA, 380–388. https://doi.org/10.1145/509907.509965

[15] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei-Fei Li. 2009. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 248–255. https://doi.org/10.1109/CVPR.2009.5206848

[16] Yue Duan, Mu Zhang, Abhishek Vasisht Bhaskar, Heng Yin, Xiaorui Pan, Tongxin Li, Xueqiang Wang, and XiaoFeng Wang. 2018. Things You May Not Know About Android (Un) Packers: A Systematic Study based on Whole-System Emulation.. In *NDSS*.

[17] Cynthia Dwork. 2008. Differential Privacy: A Survey of Results. In *Theory and Applications of Models of Computation*, Manindra Agrawal, Dingzhu Du, Zhenhua Duan, and Angsheng Li (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 1–19.

[18] Gianni D'Angelo, Massimo Ficco, and Francesco Palmieri. 2020. Malware detection in mobile environments based on Autoencoders and API-images. *J. Parallel and Distrib. Comput.* 137 (2020), 26–33. https://doi.org/10.1016/j.jpdc.2019.11.001

[19] Ruitao Feng, Sen Chen, Xiaofei Xie, Lei Ma, Guozhu Meng, Yang Liu, and Shang-Wei Lin. 2019. MobiDroid: A Performance-Sensitive Malware Detection System on Mobile Platform. In *2019 24th International Conference on Engineering of Complex Computer Systems (ICECCS)*. 61–70. https://doi.org/10.1109/ICECCS.2019.00014

[20] Daniel Gibert, Carles Mateu, and Jordi Planes. 2020. HYDRA: A multimodal deep learning framework for malware classification. *Computers & Security* 95 (2020), 101873. https://doi.org/10.1016/j.cose.2020.101873

[21] Richard Harang and Ethan M. Rudd. 2020. SOREL-20M: A Large Scale Benchmark Dataset for Malicious PE Detection. arXiv:2012.07634 [cs.CR]

[22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 770–778. https://doi.org/10.1109/CVPR.2016.90

[23] Chao-Yung Hsu, Chun-Shien Lu, and Soo-Chang Pei. 2011. Homomorphic encryption-based secure SIFT for privacy-preserving feature extraction. In *Media Watermarking, Security, and Forensics III*, Nasir D. Memon, Jana Dittmann, Adnan M. Alattar, and Edward J. Delp III (Eds.), Vol. 7880. International Society for Optics and Photonics, SPIE, 32 – 48. https://doi.org/10.1117/12.873325

[24] Cylance Inc. 2012. VirusShare.com - Because Sharing is Caring. Online. https://virusshare.com/.

[25] Young-Seob Jeong, Jiyoung Woo, and Ah Reum Kang. 2019. Malware Detection on Byte Streams of PDF Files Using Convolutional Neural Networks. *Big Data Analytics for Cyber Security* 2019 (2019), AID 8485365.

[26] Daniel Jurafsky and James H. Martin. 2009. *Speech and Language Processing (2nd Edition)*. Prentice-Hall, Inc., USA.

[27] Kaggle. 2013. Dogs vs. Cats. Online. https://www.kaggle.com/c/dogs-vs-cats.

[28] Aditya Kapoor, Himanshu Kushwaha, and Ekta Gandotra. 2019. Permission based Android Malicious Application Detection using Machine Learning. In *2019 International Conference on Signal Processing and Communication (ICSC)*. 103–108. https://doi.org/10.1109/ICSC45622.2019.8938236

[29] Kaspersky. 2019. Check if your device has been targeted by the ShadowHammer cyberattack. Online. https://shadowhammer.kaspersky.com/.

[30] Alexander Küchler, Alessandro Mantovani, Yufei Han, Leyla Bilge, and Davide Balzarotti. 2021. Does Every Second Count? Time-based Evolution of Malware

Behavior in Sandboxes. In *Proceedings of the Network and Distributed System Security Symposium, NDSS. The Internet Society.*

[31] Paul Pu Liang, Terrance Liu, Liu Ziyin, Ruslan Salakhutdinov, and Louis-Philippe Morency. 2020. Think locally, act globally: Federated learning with local and global representations. *arXiv preprint arXiv:2001.01523* (2020).

[32] Kuang-Yao Lin and Wei-Ren Huang. 2020. Using Federated Learning on Malware Classification. In *2020 22nd International Conference on Advanced Communication Technology (ICACT)*. 585–589. https://doi.org/10.23919/ICACT48636.2020.9061261

[33] Tao Lin, Lingjing Kong, Sebastian U. Stich, and Martin Jaggi. 2020. Ensemble Distillation for Robust Model Fusion in Federated Learning. *CoRR* abs/2006.07242 (2020). arXiv:2006.07242 https://arxiv.org/abs/2006.07242

[34] Alessandro Mantovani, Simone Aonzo, Xabier Ugarte-Pedrero, Alessio Merlo, and Davide Balzarotti. 2020. Prevalence and Impact of Low-Entropy Packing Schemes in the Malware Ecosystem. In *Network and Distributed System Security (NDSS) Symposium, NDSS*, Vol. 20.

[35] Hoda El Merabet and Abderrahmane Hajraoui. 2019. A Survey of Malware Detection Techniques based on Machine Learning. *International Journal of Advanced Computer Science and Applications* 10, 1 (2019). https://doi.org/10.14569/IJACSA.2019.0100148

[36] George A. Miller. 1995. WordNet: A Lexical Database for English. *Commun. ACM* 38, 11 (Nov. 1995), 39–41. https://doi.org/10.1145/219717.219748

[37] Nauman Mohammad, Tanveer Tamleek Ali, Khan Sohail, and Syed Toqeer Ali. 2018. Deep neural architectures for large scale android malware analysis. *Cluster Computing* 21 (2018), 569–588. https://doi.org/10.1007/s10586-017-0944-y

[38] Lakshmanan Nataraj, S. Karthikeyan, Gregoire Jacob, and B.S. Manjunath. 2011. Malware Images: Visualization and Automatic Classification. In *International Symposium on Visualization for Cyber Security (VizSec)*. https://vision.ece.ucsb.edu/sites/default/files/publications/nataraj_vizsec_2011_paper.pdf

[39] Modupe Odusami, Olusola Abayomi-Alli, Sanjay Misra, Olamilekan Shobayo, Robertas Damasevicius, and Rytis Maskeliunas. 2018. Android Malware Detection: A Survey. In *Applied Informatics*, Hector Florez, Cesar Diaz, and Jaime Chavarriaga (Eds.). Springer International Publishing, Cham, 255–266.

[40] Daniel W. Otter, Julian R. Medina, and Jugal K. Kalita. 2021. A Survey of the Usages of Deep Learning for Natural Language Processing. *IEEE Transactions on Neural Networks and Learning Systems* 32, 2 (2021), 604–624. https://doi.org/10.1109/TNNLS.2020.2979670

[41] Edward Raff, Jon Barker, Jared Sylvester, Robert Brandon, Bryan Catanzaro, and Charles Nicholas. 2017. Malware Detection by Eating a Whole EXE. arXiv:1710.09435 [stat.ML]

[42] José Luis Rojo-Álvarez, Manel Martínez-Ramón, Jordi Muñoz-Marí, and Gustau Camps-Valls. 2018. *From Signal Processing to Machine Learning*. 1–11. https://doi.org/10.1002/9781118705810.ch1

[43] Royi Ronen, Marian Radu, Corina Feuerstein, Elad Yom-Tov, and Mansour Ahmadi. 2018. Microsoft Malware Classification Challenge. *CoRR* abs/1802.10135 (2018). arXiv:1802.10135 http://arxiv.org/abs/1802.10135

[44] Anit Kumar Sahu, Tian Li, Maziar Sanjabi, Manzil Zaheer, Ameet Talwalkar, and Virginia Smith. 2018. On the Convergence of Federated Optimization in Heterogeneous Networks. *CoRR* abs/1812.06127 (2018). arXiv:1812.06127 http://arxiv.org/abs/1812.06127

[45] Joshua Saxe and Konstantin Berlin. 2015. Deep neural network based malware detection using two dimensional binary program features. In *2015 10th International Conference on Malicious and Unwanted Software (MALWARE)*. 11–20. https://doi.org/10.1109/MALWARE.2015.7413680

[46] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. 2017. Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization. In *2017 IEEE International Conference on Computer Vision (ICCV)*. 618–626. https://doi.org/10.1109/ICCV.2017.74

[47] Akanksha Rai Sharma and Pranav Kaushik. 2017. Literature survey of statistical, deep and reinforcement learning in natural language processing. In *2017 International Conference on Computing, Communication and Automation (ICCCA)*. 350–354. https://doi.org/10.1109/CCAA.2017.8229841

[48] Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *International Conference on Learning Representations*.

[49] Ching Y. Suen. 1979. n-Gram Statistics for Natural Language Understanding and Text Processing. *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-1, 2 (1979), 164–172. https://doi.org/10.1109/TPAMI.1979.4766902

[50] Roopak Surendran, Tony Thomas, and Sabu Emmanuel. 2020. A TAN based hybrid model for android malware detection. *Journal of Information Security and Applications* 54 (2020), 102483. https://doi.org/10.1016/j.jisa.2020.102483

[51] Duc-Ly Vu, Trong-Kha Nguyen, Tam V. Nguyen, Tu N. Nguyen, Fabio Massacci, and Phu H. Phung. 2019. A Convolutional Transformation Network for Malware Classification. arXiv:1909.07227 [cs.CR]

[52] Qi Wang, Wajih Ul Hassan, Ding Li, Kangkook Jee, Xiao Yu, Kexuan Zou, Junghwan Rhee, Zhengzhang Chen, Wei Cheng, Carl A. Gunter, and Haifeng Chen. 2020. You are what you do: Hunting stealthy malware via data provenance analysis. In *Symposium on Network and Distributed System Security (NDSS)*.

[53] Wei Wang, Vincent W. Zheng, Han Yu, and Chunyan Miao. 2019. A Survey of Zero-Shot Learning: Settings, Methods, and Applications. *ACM Trans. Intell. Syst. Technol.* 10, 2, Article 13 (2019), 37 pages. https://doi.org/10.1145/3293318

[54] Yaqing Wang, Quanming Yao, James T. Kwok, and Lionel M. Ni. 2020. Generalizing from a Few Examples: A Survey on Few-Shot Learning. *ACM Comput. Surv.* 53, 3, Article 63 (2020), 34 pages. https://doi.org/10.1145/3386252

[55] Li Yang and Junlin Liu. 2020. TuningMalconv: Malware Detection With Not Just Raw Bytes. *IEEE Access* 8 (2020), 140915–140922. https://doi.org/10.1109/ACCESS.2020.3014245

[56] Shang-Nan Yin, Ho-Seok Kang, Zhi-Guo Chen, and Sung-Ryul Kim. 2018. A Malware Detection System Based on Heterogeneous Information Network. In *Proceedings of the 2018 Conference on Research in Adaptive and Convergent Systems* (Honolulu, Hawaii) *(RACS '18)*. Association for Computing Machinery, New York, NY, USA, 154–159. https://doi.org/10.1145/3264746.3264784

[57] Lun-Pin Yuan, Wenjun Hu, Ting Yu, Peng Liu, and Sencun Zhu. 2019. Towards large-scale hunting for Android negative-day malware. In *22nd International Symposium on Research in Attacks, Intrusions and Defenses ({RAID} 2019)*. 533–545.

[58] Chen Zhang, Yu Xie, Hang Bai, Bin Yu, Weihong Li, and Yuan Gao. 2021. A survey on federated learning. *Knowledge-Based Systems* 216 (2021), 106775. https://doi.org/10.1016/j.knosys.2021.106775

[59] Xiaohan Zhang, Yuan Zhang, Ming Zhong, Daizong Ding, Yinzhi Cao, Yukun Zhang, Mi Zhang, and Min Yang. 2020. Enhancing State-of-the-art Classifiers with API Semantics to Detect Evolved Android Malware. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 757–770.

[60] Chunlei Zhao, Wenbai Zheng, Liangyi Gong, Mengzhe Zhang, and Chundong Wang. 2018. Quick and Accurate Android Malware Detection Based on Sensitive APIs. In *2018 IEEE International Conference on Smart Internet of Things (SmartIoT)*. 143–148. https://doi.org/10.1109/SmartIoT.2018.00034

[61] 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 *29th {USENIX} Security Symposium ({USENIX} Security 20)*. 2361–2378.