Android Malware Detection Using BERT

Badr Souani[1], Ahmed Khanfir[1], Alexandre Bartel[2], Kevin Allix[1], and Yves Le Traon[1]

[1] University of Luxembourg
{badr.souani, ahmed.khanfir, kevin.allix, yves.letraon}@uni.lu
[2] Umeå University
alexandre.bartel@cs.umu.se

Abstract. In this paper, we propose two empirical studies to (1) detect Android malware and (2) classify Android malware into families. We first (1) reproduce the results of MalBERT using BERT models learning with Android application's manifests obtained from 265k applications (vs. 22k for MalBERT) from the AndroZoo dataset in order to detect malware. The results of the MalBERT paper are excellent and hard to believe as a manifest only roughly represents an application, we therefore try to answer the following questions in this paper. Are the experiments from MalBERT reproducible? How important are Permissions for malware detection? Is it possible to keep or improve the results by reducing the size of the manifests? We then (2) investigate if BERT can be used to classify Android malware into families. The results show that BERT can successfully differentiate malware/goodware with 97% accuracy. Furthermore BERT can classify malware families with 93% accuracy. We also demonstrate that Android permissions are not what allows BERT to successfully classify and even that it does not actually need it.

1 Introduction

Android malware are malicious applications aiming at attacking the end-users' devices, data, money, software or third party applications and services [5]. With the democratization of smartphones, virtually everyone nowadays carries everyday a device that can access, store, and manipulate sensitive and private data. Android, being the most used smartphone operating system, is a target of choice for attackers, who create malicious applications that aim to obtain financial gains from often unsuspecting users.

In fact, new Malware are constantly being released [19], causing a constant threat and challenge for the users, the application-markets maintainers, and the security researchers.

Consequently, much effort and resources are spent to develop approaches that are able to automatically detect Malware in the unstopping flow of new applications. This includes detection approaches at the app store level such as Google PlayStore [2], or at the device level via anti-viruses [5]. Practitioners and researchers are in a constant race with the load of appearing Malware, thus, trying to detect not only previously identified Malware but also new ones. For

this purpose, they propose approaches that classify the applications into Malware or not depending on relevant suspiciousness-related components appearing in the applications. Those approaches are classified into two main categories: static and dynamic analysis techniques. The approaches based on static analysis aim at identifying Malware by parsing and evaluating the syntax of the application while the dynamic-based approaches extract information about application by instrumenting and running them in order to capture any eventual malicious/suspicious behavior of the application through its execution. Additionally, a third approach category – a hybrid one – consists of combining both static and dynamic analysis, in the hope of obtaining more and better information that could be leveraged to determine the maliciousness of a given application.

The growing interest and evolution of the machine learning techniques have engendered significant advances in the security field in general [13] and in malware detection particularly [27]. Obviously, it is more interesting and even more cost-effective [27] to save expensive human computing effort by letting the machine capture the malicious characteristics of malware, instead. In this regard, previous research has focused on defining the key-components that are the most relevant to malware detection, to better guide the learning and detection abilities of the approaches. Notably, the exotic or unexpected usage of API-calls such as the data-transfer via insecure web urls can be a determinant symptom of an eventual malicious behavior [29,23,4]. Leveraging this extra knowledge of historical malware specifications boosted the capabilities of machine learning techniques towards higher performances.

Recently Rahali et al. [24] have trained a model MalBERT based on BERT [9] – a language representation model, originally only intended for natural text processing – in order to determine whether an Android application is malicious or not by processing applications' Manifest file. More precisely, they fine-tune a pre-trained BERT model on the Manifest files of the malicious and benign Android applications included in an Android dataset collected from public resources. Their evaluation of the proposed approach shows promising results, achieving 97% of prediction accuracy. This high performance could be explained by: (1) first, the relevance of the manifest information – including the configuration and descriptive data of the application – in hinting at the presence or absence of malicious behavior in the application and (2) second, the ability of BERT in differentiating between the malicious and safe variants of these relevant components.

In this same line of research, we drive an empirical study on a large-scale dataset AndroZoo [5], where we: (1) reproduce the training and evaluation experiments of Rahali et al. [24], (2) investigate the impact of the manifest permissions on the Malware detection, (3) evaluate the xml-tags noise effect on the model performance, and finally (4) discuss the capability of the proposed approach in classifying malware by families.

Our results confirm the ones published by the authors in the original paper [24], where MalBERT achieves 97% of prediction accuracy. Surprisingly, our results show that MalBERT's representation of the Manifests is not restricted

to particular components of the Manifest. In fact, the model differentiates correctly between malware and benign applications even when fed with only the permissions, or when excluding the permissions, with almost 90% of recall and more than 93% of accuracy. Similarly, reducing the size of the input Manifests by considering only the xml values (without the tags), improves very slightly the results by 0,003% for the accuracy and 0,008% for the recall. Finally, we show that MalBERT can also be used to predict Malware families with an accuracy varying between 0,81 and 0,995.

In this paper we make the following contributions:

- A reproducibility study of MalBERT using a dataset an order of magnitude bigger (265k Android applications vs. 22k);
- An ablation study where we study the impact of different elements of the Android Manifest on the malware detection rate;
- An empirical study of the usefulness of BERT to classify Android malware into families. Results show that the approach can classify malware with 93% accuracy.

The remainder of the paper is organized as follows. In Section 2 we describe the background information necessary to understand the paper. In Section 3, we present our experimental setup. Next, in Section 4, we analyze the empirical results. We discuss the results in Section 5 and present the related work in Section 6. Finally, we conclude in Section 7.

2 Background

2.1 Malware Detection

To detect malware with machine learning, practitioners traditionally have to extract a list of features from the applications, and to represent apps as a vector. These features can be extracted using two main approaches: static analysis and dynamic analysis.

Static Analysis Static Analysis consists of analyzing an application without executing it. It can extract features such as binary signatures, the list of used libraries, or code structures. More advanced analyses generate information about the code such as a call-graph (i.e., the relationship between callee and caller functions) or control flow graphs to understand, for instance, how data flows in a function or the whole program. The power of static analysis comes from the fact that, contrary to dynamic analysis, the whole code can be reached and analyzed. This also comes with a cost in term of precision and run-time. Many static analyses have a high false positive rate since paths which cannot be executed in practice might also be analyzed. A static analysis often does not scale well and thus might take a long time to execute on realistic applications. In our experiments, we statically extract features from Android applications' manifests.

Dynamic Analysis In a dynamic analysis, the application is executed to understand its behavior. In the case of malware analysis, executions are typically performed in an isolated sandbox to prevent the malicious code from spreading to the machine running the dynamic analysis or to machines on the network. The main challenge is to find input to the application to execute as much as possible of the application's code. Extracted features could be a list of API calls or a list of DNS requests.

2.2 Android Package

Android applications are zip files whose names end with the `.apk` (Android PacKage) extension. It is a container that includes the application's code, resources, certificates, assets and a manifest. The manifest is an XML file which contains metadata describing among others the structure of the application, its name and version. Furthermore, it also includes the permissions that the application requires. Thus, a manifest is a high-level representation of an Android application. We extract features from Android applications' manifest as input for our experiments.

2.3 Transformer

More recently, researchers have tried to automate the extraction of manually-defined features, or to by-pass this step altogether.

The Transformer [28] is an architecture designed to handle sequential data. It excels in the field of NLP (Natural language processing) such as translation, question and answer, paraphrasing, and text summarization. Transformers quickly became the foundation of several impressive improvements over the previous state of the art. Introduced in 2017, Transformers have been the subject of many research papers. A Transformer consists of an Encoder and a Decoder. The Encoder, takes a sequence in input and transforms it into a continuous sequence. The Decoder then generates a sequences element by element using the previous one at each step, and the sequence generated by the encoder.

2.4 BERT

BERT [9] is an approach based on transformers and has been created for text processing tasks such as translation [34], question answering [32], text classification [26][14] or text comprehension [30]. With its impressive performance, BERT had a massive impact, and has served as the basis for many other models such as Roberta [21] which is a version of BERT model with carefully selected key hyper-parameters to improve its performance, Deberta [11] that improves BERT and Roberta models by changing its attention mechanisms and masking, or CodeBert [10] that achieves great results on both natural language code search and code documentation generation tasks. BERT (and its descendants) divides its training in two stages, Pre-training and Fine-Tuning. BERT is able

to capture high-level concepts from sentences, one of its main novelty being its use of context from sentences in both directions, forward and backward, which may explains its state-of-the-art results on NLP tasks. The **pre-training** phase consists in training the model from scratch using non-labeled data such as the Wikipedia Corpus (2500M words) [1]. The pre-training performs two "fake" tasks, i.e., tasks that have no real purpose other than to force the model to learn to capture high-level concepts:

- **Masked Language Model**: In order to exercise the model's ability to consider the context of a sentence, random words from the input sentences are masked, and BERT tries during this process to infer (i.e.,recover) the words that have been masked.
- **Next Sentence Prediction**: It consists in making BERT tries to infer whether two sentences given as input are likely to be a valid sequence of sentences. This allows BERT to learn the link between sentences, which is very useful for tasks such as questions and answers.

The **fine-tuning** phase adapts an already-trained, task-agnostic BERT model to a specific task. In practice, layers of neurons are added as output, to use the output of BERT. During the fine-tuning phase, the weights of the existing BERT model are fixed, but the weights of the newly added, task-specific layers are trained in order to obtain the desired performance on the task at hand.

This separation in two phases (pre-training and fine-tuning) is a significant advantage of BERT (and of similar approaches): The pre-training, while extremely computationally expensive, only has to be done once. The resulting pre-trained model can then be put to use in a variety of tasks, after a much less computationally expensive fine-tuning.

3 Experimental Setup

In this section, we present the experimental setup we use in our study. A high-level overall representation of the entire process is depicted in Figure 1. The process features three main steps: (1) the creation of the dataset explained in Section 3.1, (2) the pre-processing step described in Section 3.2 and (3) the fine-tuning step explained in Section 3.3.

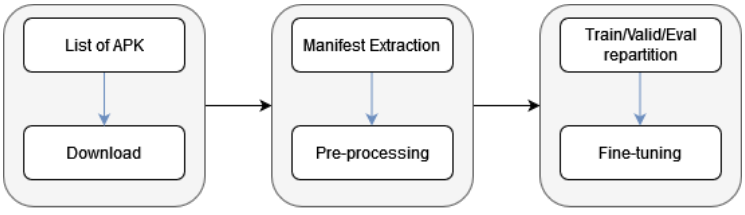

Fig. 1. Experiment representation

3.1 Dataset

In our experiments we use Android applications from AndroZoo [5]. AndroZoo is a dataset of Android apps made available to the research community, and that contains, at the time of writing, more than 19 million Android applications. All applications in AndroZoo are analyzed with several antivirus software using VirusTotal [3] in order to determine whether they are malware.

We randomly selected 265 000 Android applications released in 2019 or after, and we downloaded them from AndroZoo. The resulting dataset is composed of around 30% malware (77 768) mainly containing malware from three families[4]:

- **Jiagu** is a large family of malware. This family includes many variants that exhibit malicious behaviors such as unwanted advertisement, or *Trojan clicking*, i.e., clicking on ads without user's consent. Approximately 60% (47 522) of the malware in our dataset are of the jiagu family.
- **Dnotua** is the second largest family of malware in our dataset, representing 2% (1443) of the malware samples. Apps that are members of the Dnotua family can perform a variety of malevolent actions such as installing other apps or collecting network information.
- **Secneo** is the third largest malware family in our dataset, with 1% (674) of malware samples. Secneo apps can perform many nefarious tasks, such as sending SMS, collecting contacts, or placing phone calls.

In addition to these three families, the remaining 31% (25 182) of the malware in our dataset are either a) members of a family that contains only a small number of samples, or b) malware that do not seem to be members of a family. For our experiments, we construct training, validation, and evaluation sets, by drawing apps from the global dataset. Each experiment is conducted with a different shuffle for training, validation, and evaluation sets in order to report the most faithful values possible during evaluations. For the ground Truth of the malware detection experiments, we rely on the reports obtained from VirusTotal. For the malware family classification, we leveraged the AVclass tool [25] that can take a detection report from VirusTotal, and compute the name of the family of the sample, or a unique identifier for APKs that cannot be linked to a family.

3.2 Pre-trained Model

Since the introduction of BERT, many research teams have released their own implementation of the BERT approach, most often also accompanied by pre-trained models. In this study, we rely on a BERT model released on the TensorFlow Hub platform [5]. This model, built on top of the Tensorflow [3] library, is widely used, and follows very closely what was described in the original BERT paper [9]: It is composed of L=12 hidden layers, a hidden size of H=768 and

[3] `https://www.virustotal.com`

[4] To obtain information about malware families, we rely on the AVclass tool [25]

[5] `https://tfhub.dev`

A=12 attention heads[6]. It has been pre-trained for English on Wikipedia [1] and BooksCorpus (110M parameters)[7].

3.3 Fine-tuning

We perform two different fine-tunings for two different tasks: malware detection and malware family classification. The first tasks we investigate is malware detection. In this setting, models are fine-tuned with the aim of discriminating benign applications from malicious ones. The fine-tuning step is performed on a training-set composed of 132 500 (i.e., half the dataset) APKs from AndroZoo [5] with 30% malware. The second task we investigate, malware family classification, is different than malware detection while being closely related. The models, whose objective is to detect whether an application is part of a malware family or not, are fine-tuned with a dataset of 77 768 malware, i.e., all the malware samples of our dataset. The training, validation and test sets are distributed as 50%, 20% and 30% respectively. The training, validation and test sets are stratified, which means that each set has the same malware/goodware ratio as the whole dataset.

Regarding the parameters, the models are fine-tuned for 20 epochs with a batch size set to 32, using Adam as the optimizer function, and with a learning rate of $3e^{-5}$. All training phases and inference phases are performed on one NVIDIA Tesla V100 GPU with 32 GB of memory. As an indication, one complete experiment (i.e., fine-tuning on a training set for 20 epochs, and inferring on the test set for one given type of input) takes between 10 to 16 hours each. In addition, each complete experiment is performed ten times using a different seed (i.e., a different shuffle for Train/Validation/Test sets), in order to obtain an average of performance as representative as possible of the models.

4 Empirical Results

In this section, we investigate to the following research questions:

- RQ1 : Are the experiments from MalBERT reproducible?
- RQ2 : How important are Permissions for malware detection?
- RQ3 : Is it possible to keep or improve the results by reducing the size of the manifests?
- RQ4 : Can BERT classify families of malware?

[6] The exact model we used can be found at `https://tfhub.dev/tensorflow/bert_en_uncased_L-12_H-768_A-12/4?tf-hub-format=compressed`. We note that we also relied on the matching BERT Pre-processor available at `https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3?tf-hub-format=compressed`

[7] More information about this model as well as about the other available models of this collection can be found at `https://tfhub.dev/google/collections/bert`

4.1 RQ1: Are the experiments from MalBERT reproducible?

While reading the literature, we observed a large number of papers discussing various techniques for Android malware detection [33,15,16]. The objective was to study an Android malware detection technique using BERT. MalBERT [24], which uses BERT with as an embedding technique for manifests from APKs achieves very good results with 97% accuracy. It is not surprising to see BERT perform very well on tasks involving text such as manifests that, while being XML data, contain nonetheless mostly textual data. However, these results might be considered quite hard to believe. Indeed, obtaining such high performance with so little information—Manifest are at most a few tens of kilobits—seems at first sight both surprising and highly promising. Our objective here is therefore to first check if the manifests are really enough to represent an application in order to determine if it is malware or not.

Table 1. Results of Bert model malware detection

Model	Application	Accuracy	Loss	F_1 score
MalBERT	22 000	0.9761	0.1274	0.9547
Our study	265 000	0.970	0.183	0.949

Like in MalBERT, the BERT model we rely on was already pre-trained for English on Wikipedia [1] and BooksCorpus. The experiments are run using the previously mentioned dataset of 265 000 different manifests with 30% malware, and with 20 epochs of fine-tuning. The dataset is divided into three stratified sets as detailed above: training, validation and testing set contain respectively 50%, 20% and 30% of the dataset.

Regarding the results in Table 1, our model has a slightly lower accuracy with 0.970 opposed to 0.976 and F1 score with 0.949 in our results, and 0.9547 with MalBERT. This can be explained by the fact that their dataset consists of only 22.000 manifests with about 45% malware which is a rather different scale. MalBERT's results have therefore been successfully reproduced, it seems that it is indeed possible to identify malware using manifests.

4.2 RQ2: How important are Permissions for malware detection?

As shown above, MalBERT seems to be able to differentiate malware from benign APKs simply by using the manifests. One immediate question that follows from this observation is: What parts of the Manifest files are enabling such performance?

Permissions is the first component of a manifest we investigate the discriminating power of. Several papers [18] [8] [6] show experiments carried out on the permissions of the manifests to detect malware because this is likely to be the factor that differentiates the category to which an APK belongs. Indeed, some

permissions are more dangerous than others because they give more possibilities to the application, such as accessing sensitive information or performing actions that can alter the Android system.

In order to answer this research question, two different pre-processing were done on the manifests in order to create two new types of manifests, one with only the permissions (*Permission Only*), and one composed of manifests without the permissions (*No Permission*). Two different models were fine-tuned like before using the two new manifest types and the same parameters. The results are shown in Table 2

Table 2. Results of Bert models malware detection with pre-processed Permissions

Pre-process	Accuracy	Loss	F_1 score	Precision	Recall
Full	**0.970**	0.183	**0.949**	0.957	0.941
Permission Only	0.930	0.228	0.879	0.897	0.861
No Permission	0.967	0.193	0.943	0.952	0.933

The fine-tuned model using manifests with only the requested permissions shows an accuracy of 0.93 and an F1 score of 0.879. Permissions do allow BERT to differentiate malware from benign APKs, but permissions do not seem to be the only part of the manifest that BERT uses for malware detection as shown by the lower results of this training compared to the one using the full manifest. It can be inferred that the permissions do indeed contain information that is very relevant for a malware detector, but that the other information in the manifests also contain additional information that could be leveraged for malware detection.

Next, the results of the fine-tuning using the manifests without the permissions are 0.967 for the accuracy and 0.943 for the F1 Score. Manifests without permissions have a very slightly lower result than the originals. This proves that permissions are not necessary for BERT to get good results. One can assume that something else in the manifests allows to differentiate malware from benign applications quite accurately. Further experiments with more precise ablations will be necessary to define which part of the manifest allows BERT to operate.

4.3 RQ3: Is it possible to keep or improve the results by reducing the size of the manifests?

To determine what helps BERT to detect malware using the manifests, an intermediate step can be to remove what might be suspected of simply interfering with the learning process. For this purpose, two new variants of manifests have been created by performing a pre-processing on the manifests as before.

For the first variant, a deny list is created with words and characters arbitrarily considered as useless for learning. This list is quite short and consists of words like 'android:' which is repeated many times in the manifests, or the

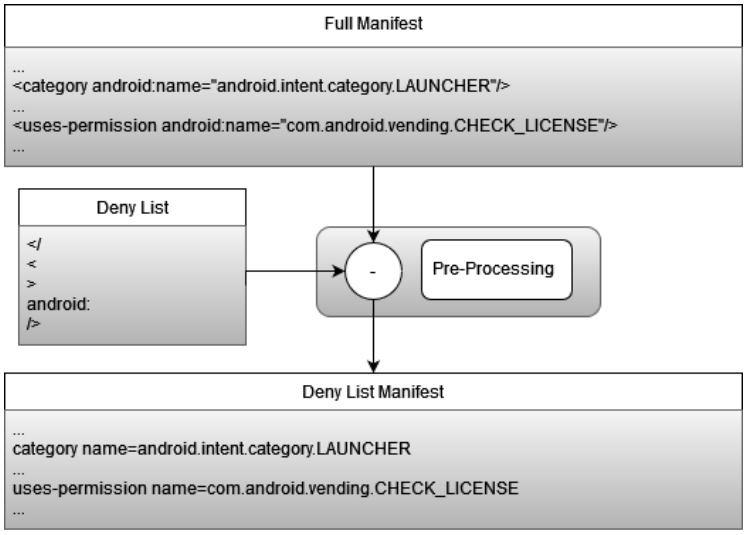

Fig. 2. Pre-process on the manifests using the deny list

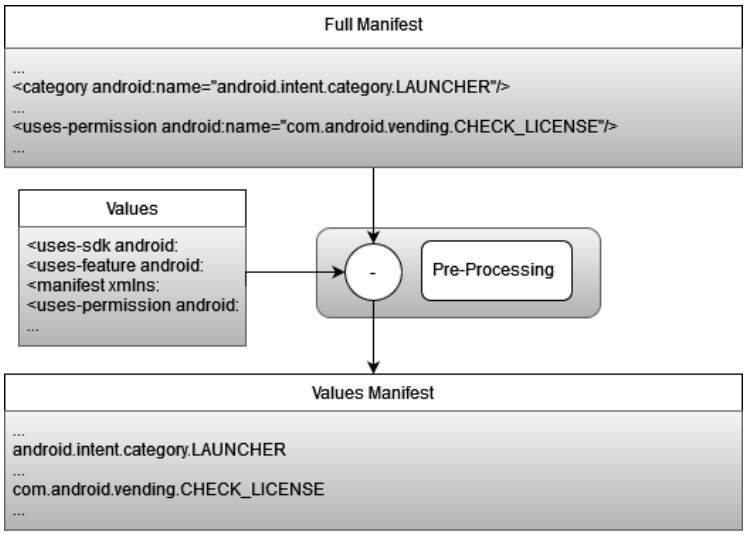

Fig. 3. Pre-process on the manifests removing tag names

less-than and greater-than signs that are heavily used to construct the XML elements. These words and letters are simply removed from the manifests and this process can be observed on Figure 2. We will refer to this manifest variant as (Deny List) For the second variant, referred to as `Values`, XML tag names are removed to keep only the XML values as shown in Figure 3. This would allow BERT to focus on what matters most. The results are presented in Table 3 and the experiments are made with the same parameters as before.

Table 3. Results of Bert models for malware detection with pre-processed manifests for Permissions and Noises

Pre-process	Accuracy	Loss	F_1 score	Precision	Recall
Full	0.970	0.183	0.949	0.957	0.941
Permission Only	0.930	0.228	0.879	0.897	0.861
No Permission	0.967	0.193	0.943	0.952	0.933
Deny List	0.972	0.168	0.951	0.961	0.942
Values	**0.973**	0.155	**0.954**	0.959	0.949

The results of the pre-precessing deny list consisting in removing the redundant words judged as being useless for the learning process allow to obtain slightly better results than the previous fine-tuning phases with an accuracy of 0.0972 and an F1 score of 0.951. This shows that reducing the "noise" indeed seems to help BERT, and that nothing necessary for its classification has been removed.

Finally, the model trained with the manifests without most of the tags but keeping the values shows the best results with 0.973 of accuracy and 0.954 of F1 score. As with the deny list, reducing the noise in the file by deleting tag names makes BERT concentrate more on what helps it to classify.

4.4 RQ4: Can BERT classify families of malware ?

It makes sense to say that BERT can detect quite accurately whether an APK is a malware using its manifest. But can BERT determine which family an application labeled as malware belongs to? In order to answer this question, the malware of the dataset have been used in order to construct a new dataset composed exclusively of malware. The experiments are carried out with the same parameters as before with the difference that the dataset is composed as explained above of 77 768 manifests. The tests and fine-tuning are carried out exclusively on the three families of malware the most present in the dataset as a consequence of the too weak presence of the other families in the dataset. These three families are Jiagu, Dnotua and Secneo with respectively 60%, 2% and 1% of presence in the dataset. The tests are performed only on the best models of each category for each family. The best model is selected by taking the one with the lowest loss value on the validation set test. The tests on the validation set are done at the end of each epoch.

Table 4. Results of Bert models Jiagu malware detection with pre-processed manifests

Pre-process	Accuracy	Loss	F_1 score	Precision	Recall
Full	0.81	0.43	0.86	0.78	0.958
Permission Only	0.758	0.522	0.827	0.734	0.946
No Permission	0.81	0.435	0.86	0.783	0.953
Deny List	0.813	0.423	0.863	0.782	0.963
Values	0.813	0.426	0.862	0.785	0.956

Table 5. Results of Bert models Dnotua malware detection with pre-processed manifests

Pre-process	Accuracy	Loss	F_1 score	Precision	Recall
Full	0.994	0.015	0.836	0.927	0.762
Permission Only	0.989	0.023	0.765	0.627	0.979
No Permission	0.995	0.013	0.865	0.902	0.831
Deny List	0.994	0.014	0.854	0.843	0.866
Values	0.992	0.019	0.822	0.719	0.958

Table 6. Results of Bert models Secneo malware detection with pre-processed manifests

Pre-process	Accuracy	Loss	F_1 score	Precision	Recall
Full	0.995	0.025	0.682	0.852	0.569
Permission Only	0.993	0.035	0.464	0.683	0.351
No Permission	0.994	0.032	0.641	0.732	0.683
Deny List	0.995	0.025	0.657	0.768	0.574
Values	0.996	0.023	0.723	0.843	0.639

Among these tables, it is important to pay attention to the F1 score which expresses more accurately the results than the accuracy, since the datasets are very unbalanced for Dnotua (Table 5) and Secneo (Table 6) unlike Jiagu (Table 4).

Table 7. Average of BERT models performance for families binary classification with pre-processed manifests

Pre-process	Accuracy	Loss	F_1 score
Full	0.933	0.232	0.793
Permission Only	0.913	0.193	0.685
No Permission	0.932	0.164	0.789
Deny List	0.934	0.154	0.791
Values	0.934	0.156	0.802

Overall, according to the F1 scores, it seems that BERT manages to classify the families: F1 scores reach on average 0.854 for Jiagu and 0.823 for Dnotua. The inferior results of Secneo with an F1 score of 633 is certainly explained by its too weak presence in the dataset, which unbalances the training and

reduces its efficiency. These results are interesting but cannot be considered as a generalization as detection from manifests can be more complex or simpler for other malware families.

Table 7 showing the average of the three family tables tells that permissions are not necessary for the detection of malware families either. This is also easily seen in Tables 4 5, and 6 which show lower results for the experiments where only the permissions are used.

5 Discussion

As shown in MalBERT [24], the results of the models trained on manifests give very good results slightly exceeding 97% accuracy for the malware/benign differentiation. The reason why this study was done is to define what exactly allows the manifests to teach BERT so well since a manifest is not enough to faithfully represent an application as manifests files are orders of magnitude smaller than applications. Moreover the approach is relatively light and easy to set up, all of the heavy lifting being already done in the BERT pre-training.

When we successfully replicated the MalBERT experiments, we expected that the manifests, once deprived of permissions would give bad results, we thought that permissions were what BERT used to differentiate malware/goodware. This information is important, since an approach relying only (or mostly) on permissions is likely to be unsuitable for real-world malware detection. Indeed, attackers can request as many permissions as they wish, and they would be quick to find combinations of permissions that are not detected as malware. But it turned out that the opposite might be true.

MalBERT seems to not use only permissions, but to also integrate in its *reasoning* other elements of the manifests as the results in Table 2 show. A further ablation study on the manifests would be interesting to understand what correlation BERT finds between the malware manifests, or the goodware ones to get its results.

It should also be noted that BERT differentiates fairly well one family from another based on the manifests, at least for the Dnotua and Jiagu families. The Secneo family does not show such good results, but this can be explained by the dataset which is rather unbalanced for this family. This remains a speculation and it is possible that BERT is simply not as effective in detecting the Secneo family as Dnotua or Jiagu. This is also true for other malware families on which further experiments would be interesting.

6 Related Work

Liu et al. [20] present different Android malware detection approaches based on machine learning. This review goes through the Android system architecture, security mechanisms, and classification of Android malware but also machine learning techniques such as data-preprocessing, feature selection and algorithms.

Similar to our paper, transformers [28] are used in MalBERT [24] in order to detect malicious software. Specifically, it uses BERT [9] based model with static analysis of Android applications to perform binary and multiclass classification. Also called MalBERT but oriented to the detection of malware affecting windows systems using BERT, *MalBERT: A novel pre-training method for malware detection* [31] uses dynamic analysis with two different datasets with more than 40 000 samples. Their results show 99.9% detection rate on their datasets and more than 98% under different robustness tests.

Malware Detection on highly imbalanced data through sequence modeling [22] also performs Android malware detection but using dynamic analysis. Furthermore, sequence activities are generated by launching the applications, and by recording their behavior. Since only a small portion of real-world applications are malicious, they recreate a real-world scenario by taking a low rate of malware in their training and testing set. Both static and dynamic analysis can lead to high performance as shown with DL-Droid [7] with deep learning systems up to 99.6% detection rate.

In a recent paper [12], the authors present an approach for malware detection using manifest permissions but without using deep-learning in contrast to us. They investigates four different machine learning algorithms, Random Forest, Support Vector Machine, Gaussian Naive Bayes and K-Means. On a test set consisting of 5243 samples, they manage to obtain results above 80%. The most effective being Random Forest with 82.5% precision and 81.5% accuracy.

CatBERT [17] is a BERT [9] model for detecting social engineering emails. They fine-tuned a BERT model with half of transformer blocks replaced with simple adapters to learn the representations of the syntax and semantics of the natural language. The model detects social engineering emails with 87% accuracy as compared to DistilBERT or LSTM which achieve 83% and 79%, respectively.

7 Conclusion

The technique used in this paper to detect Android malware and classify Android malware into families is straightforward. It uses only a BERT model and Android manifests to work. In our experiments, BERT works well for malware detection with 97% accuracy and an F1 score of 94.9%. The same goes for classification of families with on average 93.3% accuracy and 79.3% F1 score. Our experiments have shown that for malware detection, permissions alone give lower results with 93% accuracy and 87.9% F1 score, furthermore that the absence of permissions does not significantly impact the performance of the models since it obtains 96.7% accuracy and 94.3% F1 score. Finally, it is also notable that reducing the noise in the manifests used for training the models by removing redundant characters or words that are not useful for training allows BERT to obtain slightly better results. MalBERT seems to have good results and a further ablation study on the manifests would be interesting. It would help to understand what correlation BERT finds between the malware manifests, or the goodware ones to get its results.

Reproduction Package

The code used for the experiments, and the list of APKs in our dataset can be found at `https://github.com/BadrSouani/BERT_Manifest`.

Acknowledgments

This work was supported by the Luxembourg National Research Fund (FNR) (12696663). This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation

References

1. The free encyclopedia, `https://www.wikipedia.org/`
2. Google play store, `https://play.google.com/`
3. Tensorflow, `https://www.tensorflow.org/`
4. Alazab, M., Alazab, M., Shalaginov, A., Mesleh, A., Awajan, A.: Intelligent mobile malware detection using permission requests and api calls. Future Generation Computer Systems **107**, 509–521 (2020). https://doi.org/https://doi.org/10.1016/j.future.2020.02.002, `https://www.sciencedirect.com/science/article/pii/S0167739X19321223`
5. Allix, K., Bissyandé, T.F., Klein, J., Le Traon, Y.: Androzoo: Collecting millions of android apps for the research community. In: 2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR). pp. 468–471. IEEE (2016)
6. Alsoghyer, S., Almomani, I.: On the effectiveness of application permissions for android ransomware detection. In: 2020 6th Conference on Data Science and Machine Learning Applications (CDMA). pp. 94–99 (2020). https://doi.org/10.1109/CDMA47397.2020.00022
7. Alzaylaee, M.K., Yerima, S.Y., Sezer, S.: Dl-droid: Deep learning based android malware detection using real devices. Computers & Security **89**, 101663 (2020). https://doi.org/https://doi.org/10.1016/j.cose.2019.101663, `https://www.sciencedirect.com/science/article/pii/S0167404819300161`
8. Arora, A., Peddoju, S.K., Conti, M.: ¡italic¿permpair¡/italic¿: Android malware detection using permission pairs. IEEE Transactions on Information Forensics and Security **15**, 1968–1982 (2020). https://doi.org/10.1109/TIFS.2019.2950134
9. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: BERT: Pre-training of deep bidirectional transformers for language understanding pp. 4171–4186 (Jun 2019). https://doi.org/10.18653/v1/N19-1423, `https://aclanthology.org/N19-1423`
10. Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong, M., Shou, L., Qin, B., Liu, T., Jiang, D., Zhou, M.: CodeBERT: A Pre-Trained Model for Programming and Natural Languages. arXiv e-prints arXiv:2002.08155 (Feb 2020)
11. He, P., Liu, X., Gao, J., Chen, W.: DeBERTa: Decoding-enhanced BERT with Disentangled Attention. arXiv e-prints arXiv:2006.03654 (Jun 2020)
12. Jeffrey, M., Nathan, H., William, G., Ryan, B.: Machine learning-based android malware detection using manifest permissions (2021). https://doi.org/10.24251/HICSS.2021.839

13. Jimenez, M., Rwemalika, R., Papadakis, M., Sarro, F., Le Traon, Y., Harman, M.: The importance of accounting for real-world labelling when predicting software vulnerabilities. In: Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. pp. 695–705 (2019)
14. Jin, D., Jin, Z., Zhou, J.T., Szolovits, P.: Is bert really robust? natural language attack on text classification and entailment. arXiv preprint arXiv:1907.11932 (2019)
15. Karbab, E.B., Debbabi, M., Derhab, A., Mouheb, D.: Maldozer: Automatic framework for android malware detection using deep learning. Digital Investigation **24**, S48–S59 (2018). https://doi.org/https://doi.org/10.1016/j.diin.2018.01.007, https://www.sciencedirect.com/science/article/pii/S1742287618300392
16. Kim, T., Kang, B., Rho, M., Sezer, S., Im, E.G.: A multimodal deep learning method for android malware detection using various features. IEEE Transactions on Information Forensics and Security **14**(3), 773–788 (2019). https://doi.org/10.1109/TIFS.2018.2866319
17. Lee, Y., Saxe, J., Harang, R.: CATBERT: Context-Aware Tiny BERT for Detecting Social Engineering Emails. arXiv e-prints arXiv:2010.03484 (Oct 2020)
18. Li, J., Sun, L., Yan, Q., Li, Z., Srisa-an, W., Ye, H.: Significant permission identification for machine-learning-based android malware detection. IEEE Transactions on Industrial Informatics **14**(7), 3216–3225 (2018). https://doi.org/10.1109/TII.2017.2789219
19. Liu, K., Xu, S., Xu, G., Zhang, M., Sun, D., Liu, H.: A review of android malware detection approaches based on machine learning. IEEE Access **8**, 124579–124607 (2020)
20. Liu, K., Xu, S., Xu, G., Zhang, M., Sun, D., Liu, H.: A review of android malware detection approaches based on machine learning. IEEE Access **8**, 124579–124607 (2020). https://doi.org/10.1109/ACCESS.2020.3006143
21. Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., Stoyanov, V.: RoBERTa: A Robustly Optimized BERT Pretraining Approach. arXiv e-prints arXiv:1907.11692 (Jul 2019)
22. Oak, R., Du, M., Yan, D., Takawale, H., Amit, I.: Malware detection on highly imbalanced data through sequence modeling. In: Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security. p. 37–48. AISec'19, Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3338501.3357374, https://doi.org/10.1145/3338501.3357374
23. Peiravian, N., Zhu, X.: Machine learning for android malware detection using permission and api calls pp. 300–305 (2013). https://doi.org/10.1109/ICTAI.2013.53
24. Rahali, A., Akhloufi, M.A.: Malbert: Using transformers for cybersecurity and malicious software detection (2021)
25. Sebastián, M., Rivera, R., Kotzias, P., Caballero, J.: Avclass: A tool for massive malware labeling. In: Monrose, F., Dacier, M., Blanc, G., Garcia-Alfaro, J. (eds.) Research in Attacks, Intrusions, and Defenses. pp. 230–253. Springer International Publishing, Cham (2016)
26. Sun, C., Qiu, X., Xu, Y., Huang, X.: How to Fine-Tune BERT for Text Classification? arXiv e-prints arXiv:1905.05583 (May 2019)
27. Sun, T., Daoudi, N., Allix, K., Bissyandé, T.F.: Android malware detection: Looking beyond dalvik bytecode. In: 2021 36th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW). pp. 34–39. IEEE (2021)

28. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is all you need **30** (2017), https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf
29. Wu, D.J., Mao, C.H., Wei, T.E., Lee, H.M., Wu, K.P.: Droidmat: Android malware detection through manifest and api calls tracing pp. 62–69 (2012). https://doi.org/10.1109/AsiaJCIS.2012.18
30. Xu, H., Liu, B., Shu, L., Yu, P.S.: BERT Post-Training for Review Reading Comprehension and Aspect-based Sentiment Analysis. arXiv e-prints arXiv:1904.02232 (Apr 2019)
31. Xu, Z., Fang, X., Yang, G.: Malbert: A novel pre-training method for malware detection. Computers and Security **111**, 102458 (2021). https://doi.org/https://doi.org/10.1016/j.cose.2021.102458, https://www.sciencedirect.com/science/article/pii/S0167404821002820
32. Yang, W., Xie, Y., Lin, A., Li, X., Tan, L., Xiong, K., Li, M., Lin, J.: End-to-End Open-Domain Question Answering with BERTserini. arXiv e-prints arXiv:1902.01718 (Feb 2019)
33. Yuan, Z., Lu, Y., Wang, Z., Xue, Y.: Droid-sec: Deep learning in android malware detection. SIGCOMM Comput. Commun. Rev. **44**(4), 371–372 (aug 2014). https://doi.org/10.1145/2740070.2631434, https://doi.org/10.1145/2740070.2631434
34. Zhu, J., Xia, Y., Wu, L., He, D., Qin, T., Zhou, W., Li, H., Liu, T.Y.: Incorporating BERT into Neural Machine Translation. arXiv e-prints arXiv:2002.06823 (Feb 2020)