TECHNISCHE UNIVERSITÄT BERLIN

Information Systems Management Degree Program in the
Faculty of Electrical Engineering and Computer Science

Master's Thesis

Transformer-based Mobile Malware Detection

BY

Joshua-Leon Dreger

Supervisor:
Prof. Dr. Konrad Rieck

Machine Learning and Security (MLSEC)

Berlin Institute for the Foundations of Learning and Data (BIFOLD)

January 2025

Abstract

For an overview of this document, see **??**.

Provide a Thesis Abstract here (length: less than one page).

For a start, you may want to consult the concise instructions for writing the Summary Paragraph in *Nature*: https://www.nature.com/documents/nature-summary-paragraph.pdf. Moreover, consider Markus Kuhn's advice on differentiating abstract and introduction: https://www.lightbluetouchpaper.org/2007/03/14/how-not-to-write-an-abstract/.

A boilerplate scheme for an abstract is as follows: devote 25 % of the space on the purpose and importance of the research (introduction), 25 % of the space on what you did (methods), 35 % of the space on what you found (results), and 15 % of the space on the implications of the research (cf. https://writingcenter.gmu.edu/guides/writing-an-abstract).

More concrete advice for writing abstracts can be found on the website of the Writing Center of the University of North Carolina at Chapel Hill (https://writingcenter.unc.edu/tips-and-tools/abstracts/). Some useful phrases for abstracts can be found at http://dissertation.laerd.com/useful-phrases-when-writing-a-dissertation-abstract.php

Finally, you may also want to consider the excellent guide by Kent Beck on how to write good abstracts, which focuses on conference papers: https://plg.uwaterloo.ca/~migod/research/beckOOPSLA.html.

Contents

1 Introduction

Here should be a text that states:

- Problem Statement
- Objective
- Research Contribution

2 Related Work

@TODO: Summary:

2.1 Malware Detection

2.1.1 Static, Dynamic, and Network Analysis

Malware detection approaches include static, dynamic, and network-based methods [Rie24]. While this thesis focuses on static analysis, it is worth briefly noting the other approaches for context.

Static analysis examines APK features such as source code and binaries without executing the app. This makes it highly scalable and suitable for large-scale evaluations. Prior research like [Arp+14] and [Sun+24] demonstrate the effectiveness and efficiency of static approaches. However, static analysis does face challenges, including evasion techniques like code obfuscation and polymorphism.

Dynamic analysis, in contrast, observes malware behavior at runtime. Although it provides richer contextual insights, it requires significant resources and infrastructure and is vulnerable to other evasive strategies, such as sandbox detection. Similarly, network analysis examines communication patterns and data flows to identify malicious activity.

Hybrid techniques that integrate these methods aim to combine their strengths while mitigating individual limitations. These, too, face challenges such as computational complexity and integration difficulties. This thesis, therefore, focuses solely on static analysis as a scalable and efficient approach for malware detection.

2.1.2 Evasion Techniques

Malware authors often employ evasion techniques to bypass detection mechanisms [Rie24]. In static analysis, common strategies include manipulating code structures through obfuscation [Bac+18] and polymorphism [Cat+22]. Obfuscation involves transforming code into a more complex or less readable form to conceal its true purpose, whereas polymorphism allows malware to dynamically change its appearance while maintaining its core functionality. These approaches exploit the limitations of analyzing code without running it. While dynamic and network based methods address some of these weaknesses, they introduce their own

vulnerabilities, such as susceptibility to sandbox detection and reliance on communication pattern anomalies. Transformers ability to handle obfuscation and learn robust representations of static code helps mitigate some of these challenges [Roz+21].

2.1.3 Concept Drift

Concept drift refers to the gradual evolution of malware patterns, which often leads to a decline in the performance of detection models over time. Models like Transcend [Jor+17] have tackled this issue by introducing mechanisms to detect and adapt to concept drift during deployment. However, these methods are resource-intensive and may struggle to generalize across diverse datasets. Addressing concept drift remains a critical challenge for static Android malware detection and is one criteria used in this thesis.

2.1.4 Android Malware Detection

Detecting malware on the Android platform presents unique challenges and opportunities due to its open ecosystem and the high volume of apps and malware samples. The availability of many labeled datasets such as Androzoo, Drebin, Transcend???, and DexRay has driven research in this domain.

Drebin [Arp+14] represents a leap in Android malware detection, being both a dataset and an approach. As an SVM-based method, Drebin achieves remarkable results, significantly outperforming other approaches by detecting 94% of malware samples at a false-positive rate of just 1%. This demonstrated the high effectiveness of machine learning-based static Android malware detection, striking a balance between accuracy and efficiency. The Drebin dataset itself provides labeled malware samples with rich feature representations extracted from APK files, such as permissions, intents, and network addresses, making it a foundational resource in the field.

Transcend [Jor+17], on the other hand, introduced the critical concept of drift into Android malware detection. It highlights how models trained on older datasets like Drebin experience performance decay when applied to newer, evolving datasets. This evolution in malware characteristics, known as concept drift, underscores the need for adaptive retraining mechanisms. Transcend serves as a benchmark for evaluating the robustness of detection models over time and emphasizes the importance of building systems that can sustain detection accuracy in the face of changing malware landscapes.

Androzoo [All+16] is a comprehensive repository containing millions of APKs sourced from various app stores and spanning over a decade of Android app development. The dataset provides significant scale and diversity, enabling researchers to analyze trends in malware evolution and assess the effectiveness of detection methods over time. In addition to the APKs there was work done on mining metadata for the APKs, enabling further research on the landscape of Android Apps.

DetectBERT [Sun+24] is a novel transformer based approach to Android malware detection that builds upon DexBERT [Sun+23], another transformer based model designed for class level analysis of Android Smali code. Using a Correlated Multiple Instance Learning (c-MIL) framework, DetectBERT aggregates Smali class embeddings of DexBERT into an app level representation. Evaluated on the DexRay [Dao+24] dataset, which comprises 158,803 apps, DetectBERT reported competitive performance metrics, including 97% accuracy and an F1 score of 97% while also demonstrated robustness against concept drift, maintaining high detection rates when tested on newer malware samples. Additionally, DetectBERT reports to be computationally efficient, requiring only 2GB of GPU memory and achieving inference times of 0.005 seconds per app.

3 Methodology

3.1 Dataset Evaluation

The evaluation HIHI of datasets forms a critical foundation for the success of any machine learning task, especially in security-sensitive applications such as Android malware detection. A robust dataset not only enables effective training of models but also ensures generalizability across various scenarios, including concept drift. This section outlines the dataset evaluation process undertaken for this research.

The datasets that are evaluated are introduced in subsection 2.1.4. The Drebin, Transcending, and DexRay datasets are well-suited for comparison because they are specifically designed to serve as benchmarks for training and evaluating models. Each dataset is designed to train specific models and assess their performance effectively. Androzoo is notably different because it serves as a large repository for Android APKs rather than a closed dataset specifically designed for training algorithms. Additionally, it is continuously updated and expanded by the authors.

When comparing the number of APKs of each dataset, Androzoo has the most APKs with 24,855,480 (as of 25.12.2024 14:47). Since it is a repository, it will continue to grow larger. The biggest of the benchmarking datasets is Transcending, which has 259,230 available APKs. This is followed by DexRay with 159,803 APKs, and then Drebin with 129,013 APKs. These sizes are significant as they provide varied scales for evaluating model performance. In general, larger datasets are better for training and evaluating a model. However, this is only true if the dataset size correlates with novelty, meaning it introduces unique and diverse cases. For example, a dataset containing applications from various time periods or regions ensures that the model can generalize to unseen scenarios. This has been shown by [Kap+20], where they demonstrated that the performance of Transformer Decoder Models correlates with the size of the data used to train them.

Figure 3.1 shows the label distribution for each dataset. Notably, the Dexray dataset has an unusually high percentage of malware APKs. On one hand, this could help the model learn more diverse representations of malicious APKs. On the other hand, Arp et al. refer to this as a sampling bias, where "the collected data does not adequately represent the true distribution of the underlying security problem" [Arp+22]. While the actual label distribution is unknown (and likely changes depending on the model's use case), it seems unlikely that 39% of APKs are generally

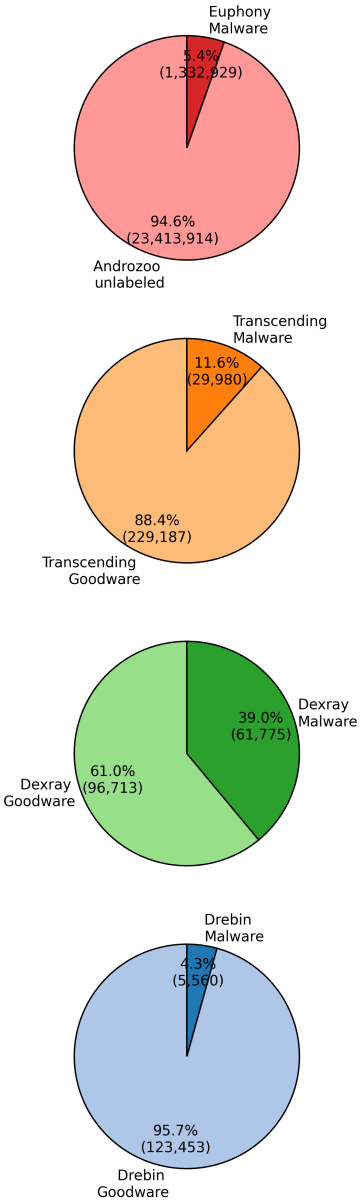

Figure 3.1: Distribution of malware and goodware samples across datasets shown as pie charts. The datasets analyzed are are ordered by size from largest to smallest. The number of APKs contained in the Dataset are shown in brackets

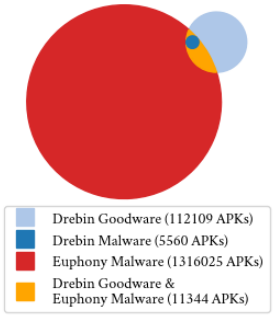

Figure 3.2: Overlap between the Drebin and Euphony datasets. The light blue circle represents Drebin, where 112,109 APKs are classified as goodware (excluding overlap). The red circle represents Euphony, which contains 1,316,025 APKs labeled as malware (excluding overlap). The dark blue circle represent those 5560 APKs, that are considered malware for both Drebin and Euphony. The intersection in orange highlights 11,344 APKs classified as goodware by Drebin but as malware by Euphony.

malware. Similarly, the Transcending dataset also has a high malware rate, with over 10% of its samples being malicious. In comparison, Androzoo and Drebin show about 5% malware APKs in their distributions. The Androzoo Malware labels were derived from virustotal[1] by Euphony [Hur+17] in 2017, which means that they are outdated as newer APKs were not evaluated.

One important aspect is that there is no clear definition of malware, which significantly impacts the analysis. This lack of a universal definition leads to inconsistencies in classification across datasets, making it challenging to validate labels and draw reliable comparisons. Without a standardized framework, each dataset might apply its own criteria. Each dataset can have different interpretations of what is considered malware, often based on criteria such as the presence of specific permissions, API calls, or behaviors flagged by antivirus software. For example, one dataset might classify an APK as malware due to aggressive adware practices, while another might only consider code executing malicious payloads. These differences make the validation of labels very difficult. One Example that shows this issue is evident when comparing Drebin with Euphony (Figure: 3.2). Euphony and Drebin have a common subset, none of the sets is a true subset of the other. However the APKs labeled as malware by Drebin are a true subset of Euphony. Additionally Euphony classifies multiple APKs as malware that were classified as goodware by Drebin. The fact that there are no APKs that are classified as malware by Drebin and Goodware by Euphony shows that Drebin applies a more narrow definition on what is considered malware. Further Overlaps of different Datasets are visualized in the Appendix (Figure: A.1).

3.1.1 Temporal Evaluation

The temporal evaluation of APKs is often based on metadata of the classes.dex component, as it contains crucial timestamp information reflecting the compilation date of the code. This metadata serves as a reliable indicator of the APK's development period, providing a foundation for chronological analyses. While this often works (Figure: 3.3), it becomes more common that this metainformation is not available. This

[1] https://www.virustotal.com/gui/home/upload

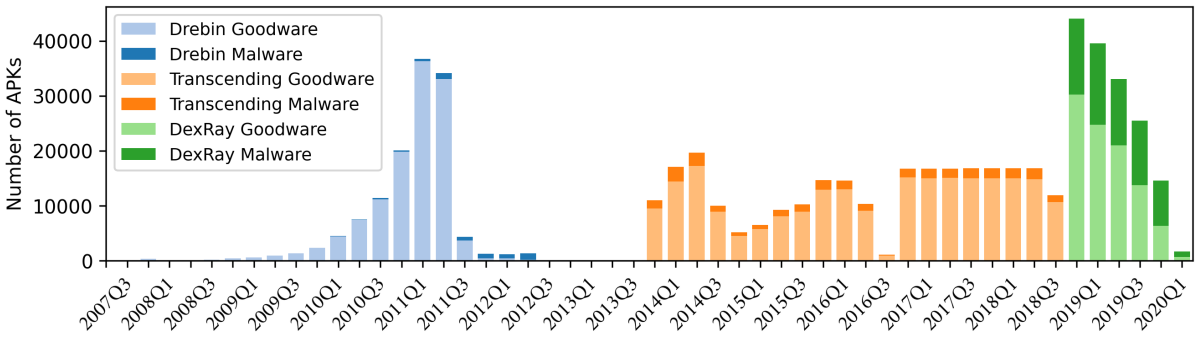

Figure 3.3: Temporal distribution of Android APKs across three datasets (Drebin, Transcend, and DexRay), categorized into goodware and malware. The Data is derived from metadata of the classes.dex file of each APK.

lack of metadata poses challenges for temporal evaluation, as it limits the ability to analyze trends and shifts in malware development over time, as seen by the temporal evaluation of the androzoo repository (Figure: 3.5)

The distribution of the .dex dates of the three datasets (Figure: 3.3) shows that the datasets are in sucession of one another. There is no temporal overlap, which explains why there is no APK present in more than one of these datasets (Figure: A.1). There is a gap between the third quarters of 2012 and 2013 where neither dataset has samples.

While the Transcending Dataset seems to be evenly distributed, the other two datasets seem to show varying volumes over time. This holds true for the overall volume but also about the evenness of the label distribution. Here Drebin and Dexray show unbalanced malware sample behaviour. This uneven distribution can affect the generalizability of findings.

One Example where concept drift is visible is, that the size of packaged APKs grows with time A.2. The three datasets show different distributions of APK sizes, where Drebin shows the smallest and DexRay the biggest APKs on average. The Plot also shows that the labels of all datasets are better distributed by the APK size, compared to the .dex date of the APK.

When evaluatig a model regarding the concept drift of malware, the Transcending dataset shows the most potential. However only considering one dataset limits the generalizability of an approach, so the evaluation on multiple datasets is even better. One Problem is, that all three datasets only contain APKs where the .dex date is available. The temporal evaluation of Androzoo 3.5 showed that this is for most modern APKs not the case. It follows that each of the datasets are Biased in their selection of of APKs and are therefore not fit for representing the True APK landscape.

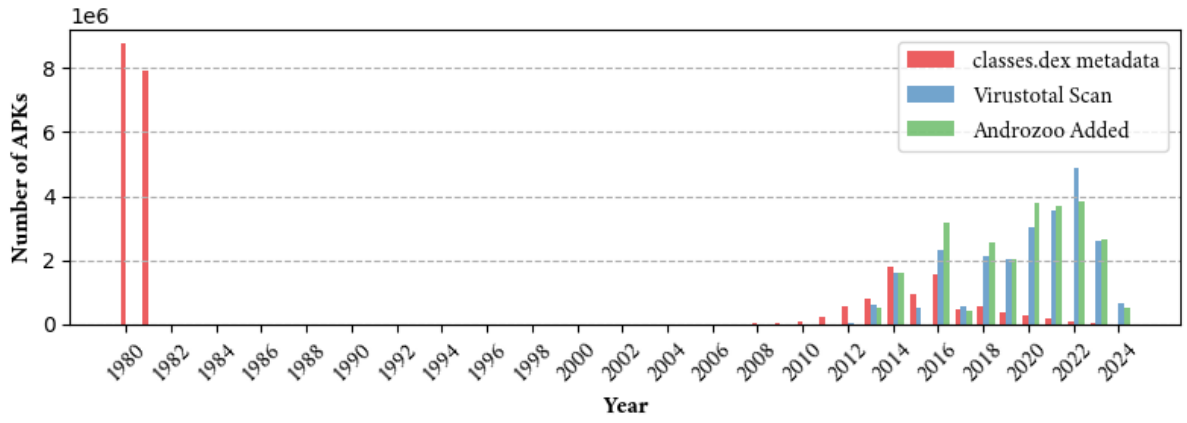

Figure 3.4: Temporal distribution of APKs based on three key attributes: classes.dex metadata, Virustotal Scan, and Androzoo Added. The red bars (classes.dex metadata) show a large spike in 1980 to 1982, likely due to incorrect or missing metadata values. The blue (year of first scan by virustotal on that APK) and green (year this APK was added to the androzoo repository) bars indicate a consistent increase in APK activity from 2010 onward, peaking around 2020 to 2022, reflecting the growing adoption of Android and corresponding malware collection efforts. The discrepancies between attributes highlight potential issues in dataset metadata accuracy and consistency.

3.1.2 Code Evaluation

When checkig for Malware, an important representation of an APK is its code. As the code describes the logic of what the App sould do, it seems logical that if the code is sufficiently evaluated the harmfulness of the APK can be derived. Especially Transformer models show promise in interpreting the code of an APK as they work very well as Large Language Models.

It follows that not just time distributions of the datasets should be evaluated, but also the distribution of code. There are two major ways of decompiling an APK into a Code representation. The classes.dex file can either be Decompiled into Java code or into more lower level smali code.

For the decompilation of the APK into Java Code, dex2jar[2] is first used to decompile the classes.dex file into a JAR file. The JAR file can then be further decompiled into plain java code. The resulting java code is on the one hand easier to interpret due to its higher level nature, however the decompiled code is prone to obfuscation that can make it hard to interpret. One example on how obfuscation can make the logic of the App more complex is by changing the package names, so that it is not possible to see the entry points that are defined in the Manifest file.

There also is an Option to decompile the classes.dex file into Smali code. Smali is the human readable representation of the Dalvik bytecode that is found in the raw classes.dex file. Smalie therefore is to Dalvik bytecode, what Assambly is to machine code. The Smali representation of an APK can be retrieved by using Apktool[3] on the classes.dex file. While the Smali representation is more accurate the the original bytecode and also is more robust to obfuscation, the readability is much harder due to the lower level and nieche nature.

[2] https://github.com/pxb1988/dex2jar
[3] https://apktool.org/

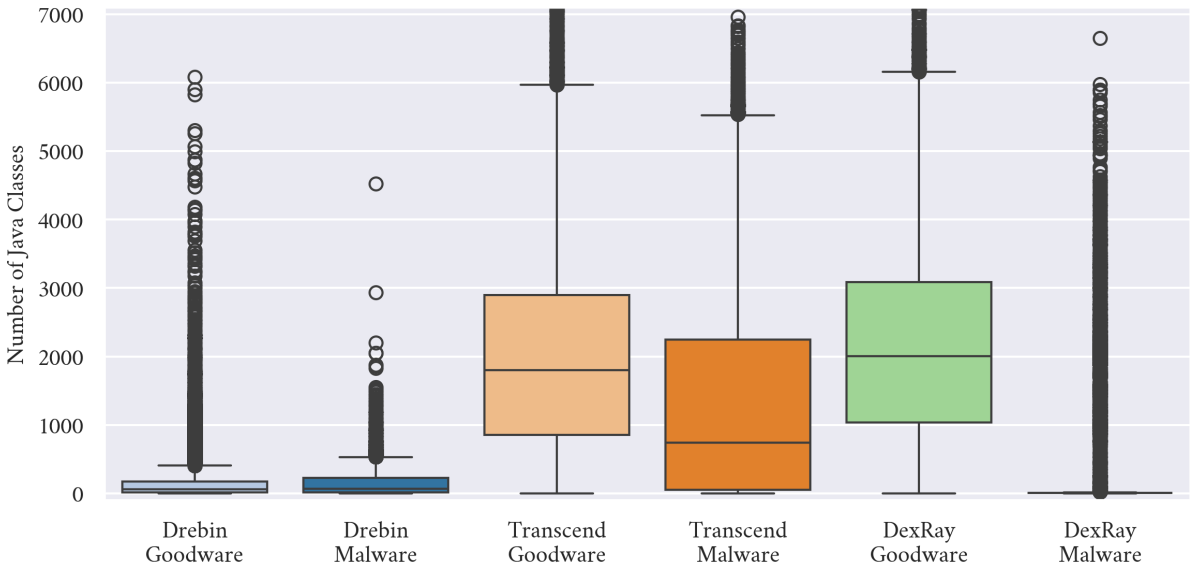

Figure 3.5: Ipsum

Table 3.1: Java Statistics Summary for DexRay, Transcend, and Drebin

Label	Q1	Q3	Median	Mean	Number of Outliers
Drebin					
Goodware	19	176	60	161.83	13243
Malware	18	224	66	176.87	506
Transcending					
Goodware	854	2899	1800	1941.68	847
Malware	52	2244	738	1312.26	996
DexRay					
Goodware	1035	3084	2007	2132.37	277
Malware	7	11	11	239.98	14308

Table 3.2: Smali Statistics Summary for DexRay, Transcend, and Drebin

Label	Q1	Q3	Median	Mean	Number of Outliers
DexRay					
Goodware	16106	41352	29546	28603.12	0
Malware	445	817	598	4113.59	13704
Transcend					
Goodware	13145	40372	25364	27234.10	1
Malware	764	25015	7057	15912.92	605
Drebin					
Goodware	331	2353	901	2275.87	14288
Malware	173	2316	805	2161.74	899

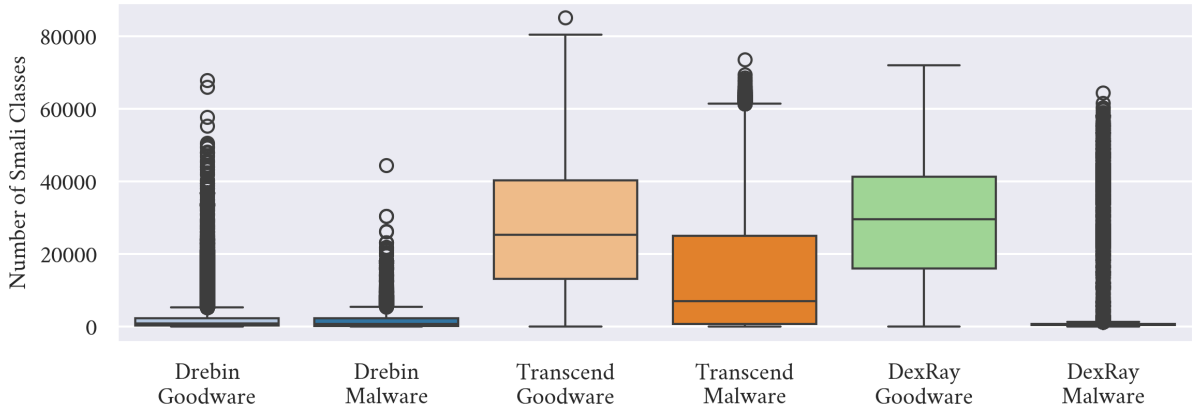

Figure 3.6: Ipsum

3.2 Baseline Creation

3.2.1 Naive Approaches

3.2.2 DetectBERT

3.3 Experimntal Setup

3.3.1 Android App Representaion

3.3.2 Model Implementation

4 Evaluation

4.1 Benchmarking Results

4.2 Experimental Results

4.3 Comparison

4.4 Interpretation

4.5 Implication

5 Conclusion

5.1 Summary

5.2 Future Work

A | Additional Plots

This appendix provides additional charts that can be helpfull fore more in depth understanding

A.1 Dataset Analysis

Figure A.1: Overlap percentages among various Android malware datasets and Google Play metadata provided by [Ale+24]. The diagonal values represent 100% overlap (self-comparison), while off-diagonal values highlight shared entries between datasets. Notable is that DexRay-, Transcending-, and Drebin Malware are subsets of Androzoo, but only partially of Androzoo malware.

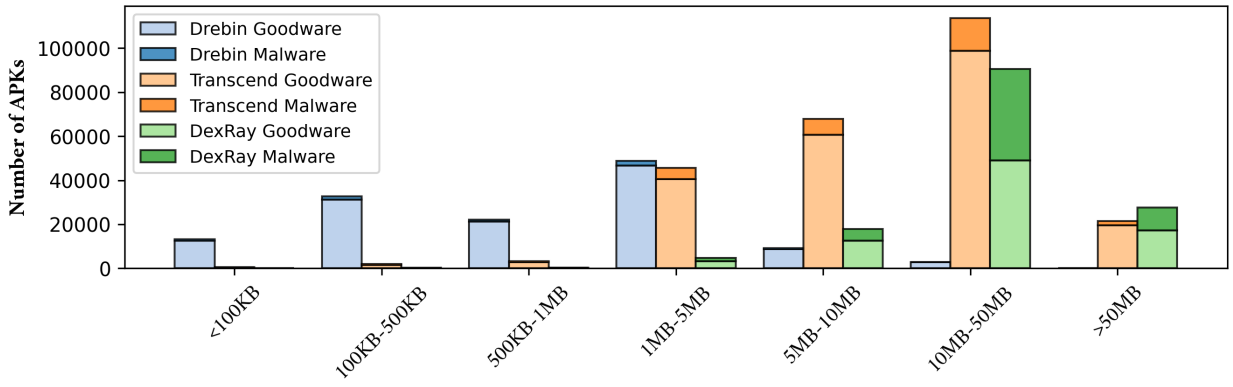

Figure A.2: Temporal distribution of Android APKs across three datasets (Drebin, Transcend, and DexRay), categorized into goodware and malware.

References

[Ale+24] M Alecci et al. AndroZoo: A Retrospective with a Glimpse into the Future. In: *2024 IEEE/ACM 21st International Conference on Mining Software Repositories (MSR)*. 2024, pp. 389–393 (page 17).

[All+16] K Allix et al. AndroZoo: Collecting Millions of Android Apps for the Research Community. In: *Proceedings of the 13th International Conference on Mining Software Repositories*. MSR '16. Austin, Texas: ACM, 2016, pp. 468–471. http://doi.acm.org/10.1145/2901739.2903508 (page 4).

[Arp+14] D Arp et al. DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket. In: *NDSS*. The Internet Society, 2014. http://dblp.uni-trier.de/db/conf/ndss/ndss2014.html#ArpSHGR14 (pages 3, 4).

[Arp+22] D Arp et al. Dos and Don'ts of Machine Learning in Computer Security. In: *31st USENIX Security Symposium (USENIX Security 22)*. Boston, MA: USENIX Association, Aug. 2022, pp. 3971–3988. https://www.usenix.org/conference/usenixsecurity22/presentation/arp (page 7).

[Bac+18] A Bacci et al. Impact of Code Obfuscation on Android Malware Detection based on Static and Dynamic Analysis. In: *ICISSP*. 2018, pp. 379–385 (page 3).

[Cat+22] C Catalano et al. Deceiving AI-based malware detection through polymorphic attacks. In: *Computers in Industry* 143:(2022), 103751. https://www.sciencedirect.com/science/article/pii/S0166361522001488 (page 3).

[Dao+24] N Daoudi et al. DexRay: A Simple, yet Effective Deep Learning Approach to Android Malware Detection based on Image Representation of Bytecode. 2024. arXiv: 2109.03326 [cs.CR]. https://arxiv.org/abs/2109.03326 (page 5).

[Hur+17] M Hurier et al. Euphony: harmonious unification of cacophonous anti-virus vendor labels for Android malware. In: *Proceedings of the 14th International Conference on Mining Software Repositories*. IEEE Press. 2017, pp. 425–435 (page 8).

[Jor+17] R Jordaney et al. Transcend: Detecting Concept Drift in Malware Classification Models. In: *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC: USENIX Association, Aug. 2017, pp. 625–642. https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/jordaney (page 4).

[Kap+20] J Kaplan et al. Scaling Laws for Neural Language Models. 2020. arXiv: 2001.08361 [cs.LG]. https://arxiv.org/abs/2001.08361 (page 7).

[Rie24] K Rieck. Lecture Slides: Machine Learning for Computer Security. https://example.com/slides. Technische Universität Berlin, Slide Deck. 2024 (page 3).

[Roz+21] B Roziere et al. DOBF: A Deobfuscation Pre-Training Objective for Programming Languages. 2021. arXiv: 2102.07492 [cs.CL]. https://arxiv.org/abs/2102.07492 (page 4).

[Sun+23] T Sun et al. DexBERT: Effective, Task-Agnostic and Fine-grained Representation Learning of Android Bytecode. 2023. arXiv: 2212.05976 [cs.SE]. https://arxiv.org/abs/2212.05976 (page 5).

[Sun+24] T Sun et al. DetectBERT: Towards Full App-Level Representation Learning to Detect Android Malware. 2024. arXiv: 2408.16353 [cs.SE]. https://arxiv.org/abs/2408.16353 (pages 3, 5).

Declaration of Authorship

Ich erkläre hiermit gemäß § 17 Abs. 2 APO, dass ich die vorstehende Masterarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Berlin, den _____ _____

Joshua-Leon Dreger