Universal Adversarial Perturbations for Malware

Raphael Labaca-Castro[†], Luis Muñoz-González[¶], Feargus Pendlebury[§‡],
Gabi Dreo Rodosek[†], Fabio Pierazzi[‡], Lorenzo Cavallaro[‡]

[†] *Universität der Bundeswehr München*
[¶] *Imperial College London*
[§] *Royal Holloway, University of London*
[‡] *King's College London*

Abstract

Machine learning classification models are vulnerable to adversarial examples—effective input-specific perturbations that can manipulate the model's output. Universal Adversarial Perturbations (UAPs), which identify noisy patterns that generalize across the input space, allow the attacker to greatly scale up the generation of these adversarial examples. Although UAPs have been explored in application domains beyond computer vision, little is known about their properties and implications in the specific context of realizable attacks, such as malware, where attackers must reason about satisfying challenging problem-space constraints.

In this paper, we explore the challenges and strengths of UAPs in the context of malware classification. We generate sequences of problem-space transformations that induce UAPs in the corresponding feature-space embedding and evaluate their effectiveness across threat models that consider a varying degree of realistic attacker knowledge. Additionally, we propose adversarial training-based mitigations using knowledge derived from the problem-space transformations, and compare against alternative feature-space defenses. Our experiments limit the effectiveness of a white box Android evasion attack to ~20% at the cost of ~3% TPR at 1% FPR. We additionally show how our method can be adapted to more restrictive application domains such as Windows malware.

We observe that while adversarial training in the feature space must deal with large and often unconstrained regions, UAPs in the problem space identify specific vulnerabilities that allow us to harden a classifier more effectively, shifting the challenges and associated cost of identifying new universal adversarial transformations back to the attacker.

1 Introduction

Universal Adversarial Perturbations (UAPs) [50] are a class of adversarial perturbations in which the same UAP can be used to induce errors in a Machine Learning (ML) classifier when applied to many different inputs. UAPs have proven to be very effective for crafting practical and physically realizable attacks in computer vision [23, 40, 50, 52] as well as for NLP tasks [72] and audio and speech classification [1, 54].

However, to the best of our knowledge, the study of universal perturbations in ML-based malware detection has not yet been explored, likely due to the difficulty of modifying real-world software while preserving malicious functionality [58]. Despite this, UAP attacks represent a very tempting opportunity from an adversary's perspective, as attackers naturally gravitate towards using low-effort/high-reward strategies to maximize profit [30, 31]. UAPs enable attackers to cheaply reuse the same collection of predefined perturbations to successfully evade detection with different types of input malware with a high probability. As well as being an attractive prospect for individual malware authors, UAPs are promising for the Malware-as-a-Service (MaaS) business model [38, 46, 74], in which service providers are interested in producing cheap universal evasive transformations at scale.

In this paper, we analyze the impact of UAP attacks against malware classifiers, revealing that they pose a significant and real threat against ML-based malware detection systems. First, we show the effectiveness of UAP attacks in the feature space against linear and non-linear classifiers is comparable to that of input-specific attacks, demonstrating the existence of a systemic vulnerability in these malware detectors. Second, our analyses in the problem space confirm this vulnerability. Thus, we propose a methodology to produce functional (real) adversarial malware that rely on UAPs. Specifically, we propose a greedy algorithm that identifies a short sequence of problem-space transformations that, when applied to a malware object, evade detection with high probability while preserving the malicious functionality.

We provide an extensive experimental evaluation across the Android and Windows malware landscape, exploring linear and non-linear ML models, including Logistic Regression (LR), Support Vector Machines (SVMs), Deep Neural Networks (DNNs), and Gradient Boosting Decision Trees (GBDTs) [39]. Our results show that unprotected models are brittle and vulnerable to our UAP attacks, even when the

attacker's knowledge about the target classifier is limited.

To defend against this threat we propose a novel method to perform adversarial training using adversarial examples created in the problem space. Adversarial training [32, 48] has proven to be one of the most promising approaches for defending against adversarial examples, however protecting against multiple types of perturbations is challenging [68]. Thus, we propose a method to perform adversarial training with UAPs by learning feature-space perturbations induced by the problem space transformations. Our approach allows us to protect against a set of manipulations used by an attacker to produce the adversarial malware, with a small decrease in the detection rate of non-adversarial malware. On the other side, our method reduces the number of adversarial examples needed to be crafted in the problem space to perform the adversarial training. We also show that, in comparison, feature-space adversarial training is not as robust against problem-space UAP attacks.

We note that we do not provide robustness against all possible adversarial ML attacks. Conversely, our defense focuses on "patching" those pockets of vulnerabilities that allow adversaries to craft realizable attacks using a predefined toolkit of transformations. While defending against *unknown unknowns* remains an open challenge, our methodology can be realistically applied to harden ML-based malware detection models against known vulnerabilities (i.e., transformations attackers rely on to evade detection). This raises the cost of creating evasive malware, as adversaries must either identify a new set of problem-space transformations or focus on input-specific attacks that may require using longer transformation chains, increasing the risk of corrupting the malware [45].

In summary, this paper makes the following contributions:

- We first demonstrate that ML malware classifiers are especially vulnerable to UAP attacks in the feature space, and we empirically show that they achieve similar effectiveness compared to input-specific attacks (§3).
- We then propose a novel attack methodology to find weaknesses in ML-based malware classifiers using UAPs. This methodology allows attackers to modify real malware in the problem space while preserving malicious semantics and plausibility (§4). We experimentally demonstrate the effectiveness of our approach by generating highly evasive Windows and Android malware variants using UAP attacks.
- We propose a novel defense to mitigate this threat based on adversarial training, using the knowledge from the evasive malware generated using our UAP attack. Our defense raises the cost for attackers and disincentives the use of powerful UAPs (§5).

We release our implementation of the attacks and the defense to the community (§8) which is designed in a modular fashion in order to better foster future research.

2 Background

We introduce major notation and pertinent background on feature-space and problem-space evasion attacks, UAPs, and adversarial training. In particular, we borrow notation from Biggio and Roli [16] and Pierazzi et al. [58].

2.1 Adversarial Evasion Attacks

In the malware domain, evasion attacks occur when an attacker modifies an object at test time to evade detection. The object can be represented in two ways: *feature-space objects* are the abstract numerical representation fed to the machine learning algorithm, whereas *problem-space objects* represent the input space, i.e., real software applications.

The feature space, label space, and problem space are denoted by $\mathcal{X}$, $\mathcal{Y}$, and $\mathcal{Z}$, respectively. Each input object $z \in \mathcal{Z}$ is associated with a ground-truth label $y \in \mathcal{Y}$. A classifier $g : \mathcal{X} \longrightarrow \mathcal{Y}$ produces a label prediction $\hat{y} = g(x)$. In order to be processed by a classifier, we must use a feature mapping function to convert it to the feature-space representation such that $\varphi : \mathcal{Z} \longrightarrow \mathcal{X} \subseteq \mathbb{R}^n$. In the software domain, the feature mapping function is not invertible nor differentiable, meaning it is not easy to find a problem-space attack with traditional gradient-based methods; moreover, with respect to the feature space, we must take into consideration several additional constraints to generate realistic, inconspicuous problem-space objects that preserve the attacker-defined semantics.

Feature-space attacks. The goal of the adversary is to transform an object $x \in \mathcal{X}$ into an object $x' \in \mathcal{X}$ in which $g(x') = t \in \mathcal{Y}$ where $t \neq y$. Hence, the adversary forces the model g to predict the wrong class for the object x'. In the malware context, we consider the case in which a malicious object is misclassified as benign.

Feature-space constraints. A set of constraints is thereby defined as Ω and consists of possible transformations in the feature-space. For example, limiting the lower and upper bounds of the features such that $\delta_{lb} \preceq \delta \preceq \delta_{ub}$ or limiting the number of features that can be modified.

Problem-space attacks. The goal of the adversary in the problem-space is to find a sequence $\mathbf{T} : T_n \circ T_{n-1} \circ \ldots \circ T_1$ where each transformation $T : \mathcal{Z} \longrightarrow \mathcal{Z}$ mutates the object z such that $g(\mathbf{T}(z)) = t \in \mathcal{Y}$ where $t \neq y$, while satisfying all problem-space constraints defined by the attacker.

Problem-space constraints. Problem-space attacks must satisfy additional constraints [58]: *available transformations*, *preserved semantics*, *robustness to preprocessing*, *plausibility*. For example, transformations in the problem-space are typically limited to addition, since removal or modification can lead to file corruption. For machine learning classifiers relying on static analysis, this is often achieved by injecting instructions that will not be executed or modifying parameters that do not affect the integrity of the file.

2.2 Universal Adversarial Perturbations

UAPs are a class of adversarial perturbations where a single perturbation applied to a large set of inputs produces errors in the target machine learning model for a large fraction of these inputs [50]. UAPs reveal systemic vulnerabilities in the target models and expose a significant risk, as they reduce the effort for the attacker to create adversarial examples, enabling practical and realistic attacks across different applications as, for example, in computer vision or object detection [19, 28, 47, 64], perceptual ad-blocking [69] or LiDAR-based object detection [20, 70]. As UAPs find patterns the target models are especially sensitive to, attackers can also use UAP attacks to craft successful and very query-efficient black-box attacks [23]. So far, UAP attacks have not been explored in the context of machine learning malware classifiers.

In our experiments, we measure the *effectiveness* of UAP attacks in terms of the Universal Evasion Rate (UER), computed over a set of inputs $\mathcal{X}$ and defined as:

$$\text{UER} = \frac{|\{x \in \mathcal{X} : \arg\max g(x+\delta) \neq y \in \mathcal{Y}\}|}{|\mathcal{X}|} \quad (1)$$

That is, UER denotes the fraction of inputs in $\mathcal{X}$ for which the classifier outputs an error when the UAP δ is added.

2.3 Adversarial Training

Adversarial training is one of the most successful and promising approaches for defending against adversarial examples [32, 48]. It involves training a model using adversarial examples crafted for each class so that the model becomes more robust to these types of inputs. The robustness gained depends on the strength and type of examples generated. Shafahi et al. [62] also proposed using UAP adversarial training to defend against UAP attacks in computer vision tasks.

However, adversarial training suffers from some limitations. When using standard adversarial training techniques, such as Projected Gradient Descent (PGD) or multi-step PGD, the cost of generating adversarial examples is very high, making them impractical for large-scale datasets—although some more specialized techniques can be used to alleviate the computational burden [61]. On the other hand, defending against multiple perturbations is challenging [68] and making the model robust to certain perturbations can facilitate evasion attacks that use different perturbations the defender did not consider during training.

3 Feature-Space UAPs for Malware

In this section, we present a motivational experiment to demonstrate that malware classifiers are especially vulnerable to UAPs crafted in the feature space—that is, without considering the set of problem-space constraints which restrict how the attacker can mutate an input object. Although in a domain such as malware feature-space attacks may be unrealistic from a practical perspective [58], this analysis exposes the systemic risk of malware classifiers to universal attacks and the importance of understanding this threat in the problem space, as we describe in the subsequent sections. To the best of our knowledge, this is the first study of the impact of UAP attacks for malware detection.

We perform an empirical evaluation of feature-space UAP attacks using two well-known malware datasets: i) SLEIPNIR [4] for Windows malware and ii) DREBIN20 [10, 58] for Android malware. SLEIPNIR consists of 34,995 malicious and 19,696 benign PE files and uses a binary feature space where each feature corresponds to a unique Windows API call, with 1 and 0 indicating presence and absence of the call, respectively. Each vector (i.e., PE representation) consists of 22,761 API calls. The DREBIN20 dataset, also a binary feature space, is presented in detail in §4.1.

For both datasets we create a random split with 60% of examples used for training and 40% for testing. Note that, without loss of generality, here we consider SLEIPNIR as a Windows representative out of simplicity, given its convenient binary feature space, although the remainder of the paper will consider a more comprehensive dataset: EMBER [6], which also includes continuous features (§4.1).

For both datasets we train a *Logistic Regression (LR)* classifier and a *Deep Neural Network (DNN)* with the following architecture: $n_f \times 1,024 \times 512 \times 1$, where $n_f = 22,761$ for SLEIPNIR and $5,000$ for DREBIN20. For the DNNs we use Leaky ReLU activation functions for the hidden layers (with negative slope equal to 0.1) and a sigmoid activation function for the output layer. We use the Adam optimizer [41] with learning rate equal to 10^{-3}, for both the LR and the DNN, and include Dropout to reduce overfitting.

We test the robustness of the LR and the DNN classifiers against input-specific and UAP attacks under perfect knowledge white box settings. For the input-specific attacks we use the attack proposed by Grosse et al. [34], which relies on the recursive computation of the Jacobian, searching at each step for the feature that maximizes the change in output in the desired direction (i.e., towards evasion). For the UAP attack we propose a method where we select the most salient features computed by the Jacobian averaged over the malware examples in the test set. We define the attacker's feature-space constraints in terms of the L_0-norm, i.e., the number of features that the attacker can modify, exploring values from $L_0 = 1$ to 20. As in Grosse et al. [34], we further assume that the attacker can only *add* features, in order to preserve malicious functionality, i.e., the attacker can only change features from 0 to 1 but not from 1 to 0. For the UAP attack, the effective change in the number of features that are set to 1 after the attack is at most L_0 for each input, i.e., some of the features for these inputs may already be set to 1, and

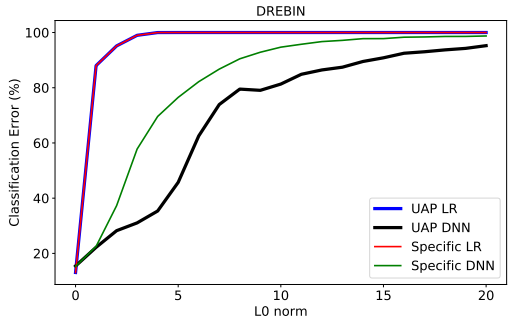

Figure 1: Android malware (DREBIN): Input-specific vs UAP white-box attacks against LR and DNN.

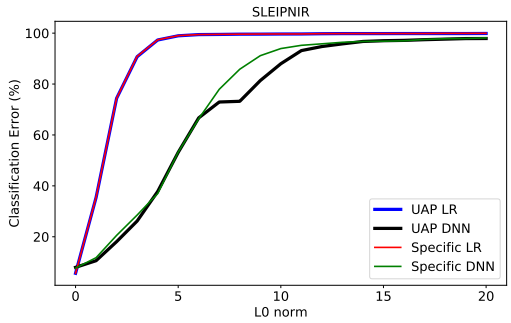

Figure 2: Windows Malware (SLEIPNIR): Input-specific vs UAP white-box attacks against LR and DNN.

thus, the UAP does not change their value.

The computation of the attacks against the LR can be simplified: for the UAP attack, we select the features with the most negative weights, i.e., we select the top-L_0 features that are most indicative of goodware. For the input-specific attacks, for each input, we also select the top-L_0 features that are most indicative of goodware and that have value 0 for that specific input.

Figures 1 and 2 show the results for the DREBIN20 and SLEIPNIR datasets, reporting the classification error of the adversarial malware at different attack strengths (including when the malware is not manipulated, i.e., $L_0 = 0$). We observe that for $L_0 = 20$, the effectiveness of both UAP and input-specific attacks is above 95% in all cases, achieving in some cases effectiveness close or equal to 100%. In other words, just by modifying 0.09% and 0.4% of the features used by SLEIPNIR and DREBIN20 classifiers respectively, we can achieve very successful attacks.

Most importantly, we observe that the effectiveness of the UAP attacks is comparable to those of the input-specific attacks, especially for the linear classifiers, where the results are almost identical. The reason is that, in the case of the LR, the features associated with the most negative weights (i.e., those indicative of goodware) are rarely present in the malware examples. Thus both UAP and input-specific attacks modify the same features in most cases.

For lower values of the L_0-norm we observe that the DNN is more robust than the LR, and that for DREBIN20, the effectiveness of the UAP attack against the DNN is slightly lower compared to input-specific attacks. However, as previously mentioned, given the very low percentage of features the attacker needs to modify to craft very successful attacks, our results show that both LR and DNN are very brittle and can be easily evaded, which is consistent with previous work [34].

For unprotected models, the extra capacity of the DNN compared to the linear classifier provides only marginal improvements in robustness that are not relevant from a practical perspective. Most importantly, our results show the importance of UAP attacks against malware classifiers: they achieve comparable effectiveness compared to their input-specific counterparts, but pose a significantly higher risk, as the same perturbation generalizes across many malware examples.

Our results suggest that there are systemic vulnerabilities in machine learning malware classifiers that attackers can leverage to craft very effective UAPs capable of evading detection regardless of the malware they are applied to. This reduces the cost for the attacker to generate adversarial malware examples at scale. These results justify the attack methodology considered in the following sections, where we show it is also possible to generate very effective UAP attacks in the problem space, which pose a significant and real threat.

4 Problem-Space UAPs for Malware

Motivated by the results of feature-space UAP attacks in §3, here we show the feasibility of generating *problem-space* UAPs to realize real-world evasive malware. We aim to answer the following research questions:

RQ1. Is it possible to generate UAPs that are *effective* at evading *malware classifiers*? (§4.3)

RQ2. Is it possible to find effective UAPs when the attacker has only *limited knowledge*? (§4.4)

To this end, we consider two different experimental settings (§4.1): first we consider an attack against an Android classifier in which the attacker is relatively unconstrained (§4.3), and secondly, an attack against a Windows malware classifier in which the attacker is more constrained, with limited knowledge and a more opaque set of available transformations (§4.4). An overview of the methodology for generation of problem-space UAPs is discussed in §4.2.

These different settings help us explore the nuances of physically transforming binaries with UAPs, as well as helping us gauge how realistic the threat of UAPs really are—across different domains.

4.1 Experimental Setting

Here we outline the threat model and datasets we consider which act as a foundation for the domain-specific attacks in §4.3 and §4.4.

4.1.1 Attack Scope and Objectives

While recent work has shown that feature-space UAPs can be employed in attacks at training time, such as backdoor poisoning attacks [77], here we focus solely on the test phase of the machine learning pipeline. Specifically, we focus on *evasion attacks* (§2.1) in which the attacker modifies objects at test-time in order to induce targeted misclassifications.

We envision a profit-motivated adversary such as a Malware-as-a-Service (MaaS) provider [38, 46, 74] with two objectives:

O1. They aim to *maximize* the amount of malware that can be made undetectable, increasing revenue.
O2. They aim to *minimize* the cost of making a single malware undetectable, reducing expenditure.

From these objectives it is clear why UAPs are a natural choice: UAPs are *scalable*, amortizing the cost of generating a perturbation over the total number of evasive malware that it produces. To quantify the success of these objectives, we use the *Universal Evasion Rate* (UER) to measure the universality of each perturbation as defined in Equation (1).

4.1.2 Attacker Knowledge

We consider two different levels of attacker knowledge. Following Biggio and Roli [16] and Carlini et al. [21] we define knowledge in terms of training data $\mathcal{D}$, feature set $\mathcal{X}$, algorithm g, and parameters/hyperparameters $\boldsymbol{w}$.

Perfect Knowledge (PK). In this setting $\theta_{PK} = \{\mathcal{D}, \mathcal{X}, g, \boldsymbol{w}\}$, the attacker has full knowledge of the learner and its hyperparameters, or otherwise, the ability to unconditionally query the target model and receive soft labels. This setting facilitates the strong attack in §4.3 that uses a greedy algorithm to generate the strongest possible UAP.

Limited Knowledge (LK). In this setting $\theta_{LK} = \{\hat{\mathcal{D}}\}$, the attacker is able to approximate the input data distribution in order to validate the strength of generated UAPs and choose the strongest of them to apply during the test-time attack. Here we assume that the attacker is also able to query the target model to receive soft labels as in past black box attacks [23]. Alternatively, queries can be made to a surrogate learner if the attacker has additional knowledge of the learning algorithm and feature set, i.e., $\theta_{LK} = \{\hat{\mathcal{D}}, \hat{\mathcal{X}}, \hat{g}, \hat{\boldsymbol{w}}\}$.

When considering a Malware-as-a-Service scenario and UAPs, the Android ecosystem is naturally appropriate to use as a perfect knowledge setting, especially as the attacker has access to bytecode, on which they can perform more detailed injections at scale. Conversely, the Windows domain is more interesting to evaluate as a limited knowledge setting in which a service provider would only have access to binaries and where it may be difficult even to discern symbols and sections [56]. Hence, we focus on the Android domain for the Perfect Knowledge setting, whereas for the Limited Knowledge setting we focus on Windows while presenting some results on Android as well.

4.1.3 Attacker Capabilities

Generally we assume the attacker is able to add, remove, or modify features arbitrarily, so long as the resulting perturbations correspond to a realizable, functioning input object. However, we do not assume the attacker has access to the original source code, as they may be a third-party operating on behalf of the malware author (e.g., a MaaS provider).

Otherwise, capabilities are domain specific, largely relating to the set of *available transformations* that the attacker has access to. In our Android attack, the attacker knows which feature-space perturbations will be induced by which problem-space transformations, giving the adversary greater control over how to mutate the binary effectively. In our Windows attack, the attacker is more constrained as the effect a certain transformation (e.g., 'UPX pack') will have on the input binary cannot be calculated a priori. We refer to §4.3 and §4.4 for the details of capabilities assumed by our Android and Windows attacks, respectively.

We do not put hard limits on the size of the perturbation in terms of L_p norm, as these have been shown to be inappropriate for formulating problem-space attacks [58]—however, we note that larger perturbations often correspond to larger transformation sequences which increase the risk of corrupting the input malware.

4.1.4 Target Models

To avoid coming to conclusions which are only specific to a particular domain, we explore attacks against malware detectors for both Windows and Android.

DREBIN Classifier. We consider DREBIN [10], an Android malware detector which can achieve state-of-the-art performance in the presence of concept drift if retrained with incremental retraining [57]. DREBIN [10] uses a linear Support Vector Machine (SVM) as the underlying classifier. For the SVM regularization hyperparameter we use $C = 1$.

EMBER Classifier. We consider a state-of-the-art Windows malware detector proposed by Anderson and Roth [6] which uses Gradient Boosting Decision Trees (GBDT) trained using the LightGBM library [39] and default hyperparameters of 100 trees with 31 leaves each.

4.1.5 Datasets

We present the main characteristics of the two datasets from related work that we use in our experiments.

DREBIN20. This Android malware dataset by Pierazzi et al. [58] consists of 152,632 benign and 17,625 malicious apps from AndroZoo [5], following the guidelines of TESSERACT [57] to avoid spatial bias. The apps are dated between Jan 2017 and Dec 2018 inclusive. The apps are embedded in the DREBIN [10] feature space abstraction, i.e., a binary feature space in which Android components (activities, permissions, URLs, services, etc) are represented as either present or absent. The apps have been labeled using a common criteria [49, 57] in which apps are labeled as *malicious* if they are detected by 4+ VirusTotal AV engines and *benign* if they are completely undetected.

EMBER. This Windows malware dataset by Anderson and Roth [6] consists of features extracted from 400,000 benign and 400,000 malicious PE files (as well as 300,000 unlabeled examples which we discard). The remaining apps are mostly dated between Jan 2017 and Dec 2017 inclusive with ~4% predating 2017. The examples have been labeled as *malicious* if they are detected by 40+ VirusTotal AV engines and *benign* if they are completely undetected. We augment the original EMBER dataset with 1,100 binaries, classified as malicious by the target model, to which we can apply problem-space transformations. The EMBER feature space has three broad types of features related to parsed features (e.g., file size, header information), format-agnostic histograms (e.g., byte-value/entropy histograms), and printable strings (e.g., character histograms, average length, URL frequency). Note that unlike DREBIN20, EMBER includes continuous features.

After explaining our methodology in §4.2, we discuss the dataset splits used in our experiments for DREBIN20 [58] and EMBER [6] in §4.3.2 and §4.4.2, respectively.

Moreover, we note that while the original labeling criteria [6, 58] discard 'difficult to classify' *grayware* with between 1 and 3 (Android) and 1 and 39 (Windows) AV detections, which could result in sampling bias [11], this would only be to the advantage of the classifiers under attack (i.e.,, it is harder for an attacker to evade this classifier). This is also true for the potential spatial bias present in the original EMBER dataset [6]. While the overall performance of the classifiers as reported in their respective original works could be inflated, in this work we are focused on the *relative degradation* in performance, before and after attacks or defenses are applied, which are unaffected.

4.2 Methodology for Generating UAPs

Generating UAPs that can be used with real-world malware is significantly more challenging than generating UAPs in the feature space (§3). In order for a UAP to be successfully applied to real-world malware, there must exist some inverse mapping from the UAP feature vector back to the "problem space"; i.e., there must exist some chain of real-world transformations which is capable of inducing the feature-space change in the chosen object. While the complexity of software means that how these real-world transformation chains are found is largely specific to the given domain, here we outline a number of common components that make up our overall methodology.

Available transformations. We augment the given threat model with the *available transformations* problem-space constraint (§7). This represents the specific *toolbox* that the attacker has access to, e.g., a set of gadgets to inject (§4.3) or a tool for performing binary mutations (§4.4). Formally we define it as a set of domain-specific problem-space transformations where each transformation is a function $T : Z \to Z$ that mutates a problem-space object $z \in Z$ into $z' \in Z$. This set is analogous to an *action space* in reinforcement learning [73]. The specific transformations are different between the Android and Windows domain, and are discussed more in detail in the appropriate experimental sections.

UAP search. Next, we perform a *greedy search* for a chain of transformations $\mathbf{T} = T_n \circ T_{n-1} \circ \ldots \circ T_1$ which can be universally applied to a set of true positive malware in order to flip their labels to benign—this chain is the problem-space equivalent of a UAP. The chain is constructed such that each new transformation aims to maximize UER, however whether this search can be feature/gradient-driven or problem-driven depends on the set of transformations itself. In order to avoid experimental bias (e.g., data snooping) we conduct this search on an *exploration set*, a partition of the training data [11]. Note that we avoid splitting the dataset temporally [57] in order to evaluate the attacks in the *absence of concept drift*, as performance degradation induced by the evolution of malware over time may lead us to overestimate the UAP success rate.

Feature space analysis. Finally we evaluate the effectiveness of the discovered UAPs on a separate test set in terms of the UER. To understand the effect that the UAPs have on the target classifier—and better understand systemic weaknesses in the model—we analyze the feature-space perturbations induced by the problem-space UAPs.

4.3 Perfect Knowledge Setting

Here we aim to answer **RQ1**: given a strong attacker, can UAPs be produced that are effective against a malware classifier? For this we consider the PK threat model (§4.1) and build on the state-of-the-art, problem-space attack proposed by Pierazzi et al. [58] that targets Android malware detectors.

4.3.1 Available Transformations for Android

We adapt the procedure from Pierazzi et al. [58] which builds on *automated software transplantation* [14]: code gadgets

are first extracted from a corpus of benign apps and then injected into a host malware until evasion occurs. Gadgets are extracted recursively to preserve dependencies up to a certain distance to improve plausibility. Although this induces *side-effect features*—extra features which may help or harm the evasion effort—it ensures that the injected gadgets are less conspicuous than, for example, no-op API calls [60] or unused permissions [34]. We extract 1,395 problem-space gadgets, based on features considered important with respect to benign examples in our exploration set, to obtain the final set of available transformations $\mathcal{T} = \{t_0, \ldots, t_{1394}\}$, where t_i denotes the injection of gadget i into a given malware. Note that none of the transformations are capable of removal, only addition (i.e., setting a feature value to 1).

4.3.2 Target DREBIN Model Baseline

The target DREBIN model is trained using DREBIN20 (§4.1). We use a random stratified split with 33% hold out for testing, partitioning the dataset into 101,596 and 50,041 examples for training and test, respectively.

We aim to to discover UAPs which are effective against the test data without overfitting, so we further divide the training set to use 90% of the examples (91,436) for the actual training and 10% (10,160) as the *exploration set*, set aside for the UAP search. This also simulates our MaaS scenario in which an adversary is interested in *reusing* UAPs on future examples which they may not yet have access to. As our *adversarial test set*, we consider all true positive malware examples detected by the trained classifier (4,503 examples). On the non-adversarial (clean) test data the model achieves an AUC-ROC of 0.981 and 0.855 TPR at 1% FPR.

4.3.3 UAP Search

In Pierazzi et al. [58], gadgets are selected greedily based on their total benign contribution (i.e., considering side effects) and added until the decision score of the host malware is sufficiently benign. Here we alter the search strategy to consider the UER across all true positive malware examples. We iteratively apply all possible transformations, at each step selecting the one maximizing UER across all true positive malware in the *exploration set*, until either the maximum length for the transformation sequence $\mathbf{T}$ is reached, 100% UER is reached, or no remaining transformations can increase UER. We consider a maximum sequence length of ten.

4.3.4 Results Analysis

The strongest UAPs that we discover using the exploration set produces 4,413 evasive variants on the test set after a single transformation (98% UER) and achieves 100% UER after two transformations. As the attack seems very strong, achieving 100% UER long before the maximum chain length of 10 is reached during exploration, we next investigate the strength

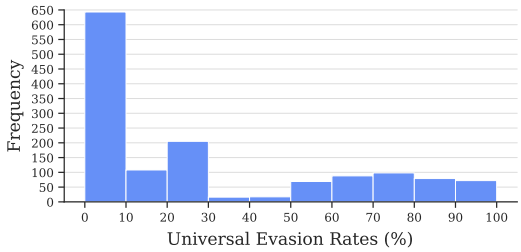

Figure 3: Histogram of Universal Evasion Rates (UERs) induced by each available *individual* problem-space transformation targeting the linear DREBIN Android malware detector.

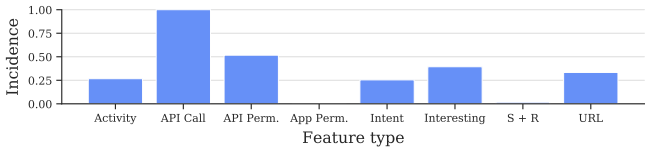

Figure 4: Relative incidence of feature perturbations, grouped by type, induced by the most effective individual transformations (UER $\geq$ 90%) targeting DREBIN.

of each transformation individually, as shown in Figure 3. While 46% of the transformations achieve less than 10% UER, 29% achieve UER of 50% or greater, with 5% of the transformations being at least 90% effective.

We next examine the nature of the feature-space perturbations induced by these strong transformations, to better understand the weaknesses of the classifier. Figure 4 shows the relative incidence of features, grouped by feature type, across the highly effective transformations (i.e., with UER $\geq$ 90%). The most common feature types perturbed by the UAPs are related to API calls, with API calls perturbed by all transformations, API-related permissions perturbed by half, and a special category of "interesting" API calls being the third most common. However, the *individual* features which occur consistently across all of the top transformations are Activities, such as `activities::CloudAndWifiBaseActivity` (which is present in all but two of these transformations).

Although we reiterate that L_p norm constraints on the perturbation are not appropriate for problem-space attacks as the object can be modified arbitrarily so long as the problem-space constraints are not violated [58], it is still worth examining the size of the L_0 distortion induced by each transformation given how strong they appear to be individually.

Figure 5 shows the distribution of L_0 perturbation sizes, with a mean and median of 18.5 and 19, respectively. To provide perspective, the L_0 perturbation induced by the strongest transformation chain is 19; the mean and median L_0 norms of the DREBIN20 dataset are 50 and 49, respectively.

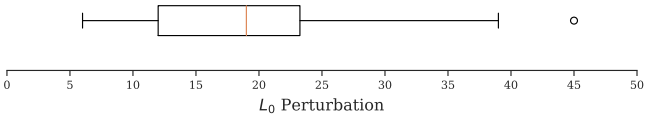

Figure 5: Distribution of L_0 norm perturbations (i.e., number of changed binary features) induced by the most effective individual transformations (UER $\geq 90\%$) targeting DREBIN.

4.4 Limited Knowledge Setting

After demonstrating the feasibility of problem-space UAPs in the PK setting, here we look to see how effective UAPs might be in a setting where the attacker is more constrained (**RQ2**). For this we consider the LK threat model (§4.1).

Figure 6 shows the results from a naive LK attack against the Android classifier, in which **T** is constructed by selecting gadgets at random. Each line depicts the UER produced by one of 1,000 transformation chains, tested at each stage of construction. The attack still appears to be extremely potent, with chains at length 5 achieving a median UER above 90%.

While clearly effective, the Android domain is naturally more amenable to powerful attacks. The attacker is able to directly manipulate the bytecode, with established program analysis tools such as Soot [71] and FlowDroid [12] making specific alterations relatively straightforward. Additionally, the toolkit of transformations in the Android attack simplifies the search, as the UER is monotonic with respect to gadget injection—there is no risk of a transformation reducing the evasiveness of the transformation chain.

A more challenging domain is that of Windows PE binaries, which are more prone to breaking runtime semantics during problem-space transformation than Android apps, due to lack of access to source code. Because of this fragility, a common semantic-preserving attack is to simply append random bytes to the end of the binary [43, 63]. However, these transformations may be detected and removed before classification. Conversely, using more sophisticated transformations increases the risk of disrupting the original malicious semantics and transformations that subtract malicious features (such as packing or compression) may equally obfuscate benign features. In the remainder, we borrow a variety of Windows problem-space transformations from related work, and use them to build problem-space UAPs attacks.

4.4.1 Available Transformations for Windows

We use the transformations proposed by Anderson et al. [7] and implemented by Labaca-Castro et al. [44]. The transformations are byte-level modifications which can be divided into three categories; *i) inclusion*: adding a new unused section, appending bytes with random length to the space at end of sections or end of the file, adding unused functions to the import address table (similar to Hu and Tan [35]); *ii) modification*: renaming sections using alternatives parsed from

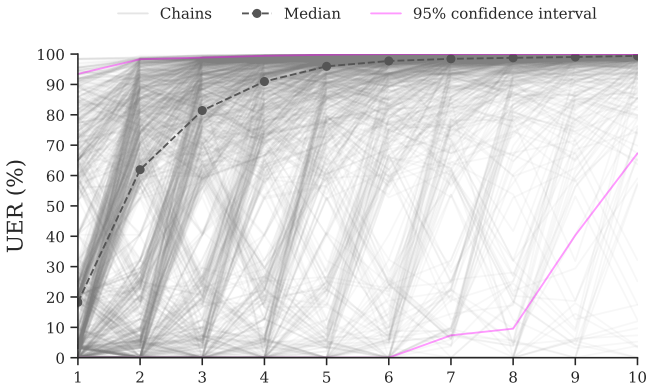

Figure 6: Limited Knowledge (LK) attack against linear DREBIN Android malware classifier. Universal Evasion Rates (UER) for 1000 random transformation chains up to length 10. Relatively few transformations are required to achieve a high UER, highlighted by the median at each stage.

Table 1: Set of available transformations for Windows PEs.

t_i	Explanation	t_i	Explanation
t_0	Append overlay	t_5	Remove signature
t_1	Append imports	t_6	Remove debug
t_2	Rename section	t_7	UPX pack
t_3	Add section	t_8	UPX unpack
t_4	Append section	t_9	Break optional header

benign files, manipulating the checksum, debug, and signer info in the header; *iii) compression*: packing or unpacking files using UPX [67] with random compression rates. Table 1 summarizes the complete set of transformations.

4.4.2 Target EMBER Model Baseline

The target model is trained on the EMBER [6] dataset (§4.1) using 300,000 malicious and 300,000 benign examples. The remaining 100,000 malicious and 100,000 benign examples comprise the clean (non-adversarial) test set. As the original dataset contains only extracted features, we augment the dataset with 1,100 malicious binaries downloaded from the VirusTotal [33] and VirusShare [25] repositories, to facilitate the problem-space attacks. All of these samples are successfully detected by the trained model. This set is partitioned in two to obtain an *exploration set* of 100 samples used to search for UAPs, and an *adversarial test set* of 1,000 samples used to validate their effectiveness. On the clean (non-adversarial) test data the model achieves an AUC-ROC of 0.999, with 0.981 TPR at 1% FPR.

4.4.3 UAP Search

To search for problem-space UAPs, we apply each of the ten transformations to each of the samples in the exploration set. After each transformation, we execute the malware in

a sandbox to verify it has not been corrupted. To maximize effectiveness of the UAP, we observe that a single transformation is unlikely to result in a large number of evasions (compared to the strength of a single gadget injection in the Android setting), so relying on hard labels to maximize UER directly may not give enough information to guide the search as UER is likely to be 0% for the first round. Given this, we average the prediction confidence output by the model across the modified examples. The transformation that minimizes the average confidence is selected, and fixed in the first position of the transformation chain. In the subsequent rounds, the same procedure is used to search for the next best transformation, and so on. If soft labels are not available, the cartesian product of transformation chains can be calculated until a length is reached which produces a usable UER signal, although the transformation cost would be exponential with respect to the length of the chains. Failing this, sequences could be sampled at random.

The number of search rounds r is a product of the size of the exploration set E, desired transformation chain $\mathbf{T}$, and available transformations, $\mathcal{T}$—i.e., $r = |E| \cdot |\mathbf{T}| \cdot |\mathcal{T}|$.

This process continues until the maximum length of $\mathbf{T}$ is reached. As per Labaca-Castro et al. [45] we stop at length 4 since minimal mutations are desired to ensure *plausibility*.

4.4.4 Results Analysis

Here we analyze the effectiveness of the most successful candidate UAP chains after applying them to the adversarial test set. We examine six candidates: three of the top scoring chains, two other high scoring chains with shorter chain lengths, and a low scoring chain to investigate the less successful sequences.

Figure 7 shows the success rates for the six chains. The most successful chain (t_7, t_1, t_6, t_4) produced 298 evasive variants from 992 [1] files (30% UER). Chains (t_7, t_1, t_6, t_8) and (t_7, t_1, t_6) both produced 288 evasive variants each (29% UER), yet the latter requires only three perturbations rather than four. This indicates that in the first two chains, the final transformation does not move the example toward the decision boundary much further than the initial three. An even shorter chain, (t_7, t_1), also achieves a relatively high evasion rate of 287 evasive variants (29%) using only two transformations. As the most common initial transformation, a single application of t_7 produces 270 evasive variants (27.2%), which clearly indicates how susceptible the model is to UPX packing. This is likely due to the model paying special attention to the structure of the binary file and therefore being more sensitive to the changes caused by high-compression ratio packers. These results support prior work which found that the distributions of header information for benign and malicious samples packed with UPX are very similar [2].

[1] Eight files return either parsing errors or are not detected by the baseline model and were therefore dismissed.

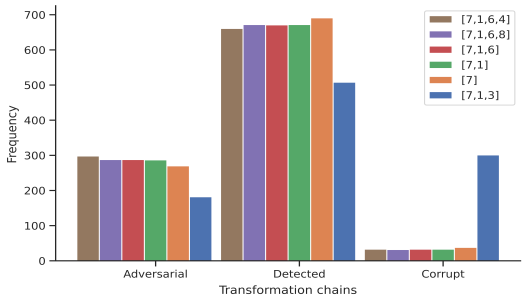

Figure 7: Reported best candidates for UAP chain in the validation set. The UAP (brown) is identified as (t_7, t_1, t_6, t_4) and leads to successful adversarial examples in 298 files whereas (t_7, t_1, t_3) is used as control to show how specific transformations at the end of the chain can drastically increase corrupt samples and hence limit the success of the chain.

On the other hand, successful initial transformations do not guarantee a successful chain. For example, appending transformation t_3 to the highly effective vector (t_7, t_1) causes the evasion rate to decrease by almost 40%. This is because the combination (t_7, t_1, t_3) produces almost 10 times more corrupt examples than the alternatives, with 301 non-functional files compared to an average of 33 across other chains.

The UAP search provides us with very useful insight about how to bypass a classifier in the problem space using a limited transformation toolkit. However, to better understand why some transformations have such a good impact on decreasing the confidence rate of the model, we analyze the feature-space perturbations induced by the chains.

We observe that applying a single transformation results in 27.7% of features being modified, on average. This number does not increase significantly given longer transformation chains. While this may be caused by the analyzed candidate chains being relatively similar (as the greedy search discards most poor candidates), the most important features for the classifier seem to be perturbed regardless of the chain length.

Furthermore, we analyze the average value of the change ('delta variation') for each feature, for each of the selected candidate chains. As shown in Figure 8, despite the individual success rates of each candidates, the features with high variation appear to be uniform across all chains. However, we do see a distinct difference in the values of features 800 through 1,000 for the less effective candidate (t_7, t_1, t_3). This feature group relates to information about the binary's sections, such as names and sizes, which appear to correspond to the final transformation t_3, 'add section'.

5 Evaluating Robustness to UAPs

After demonstrating the brittleness of ML malware classifiers against feature- and problem-space UAPs, we now evaluate strategies to improve the resilience of machine learning mod-

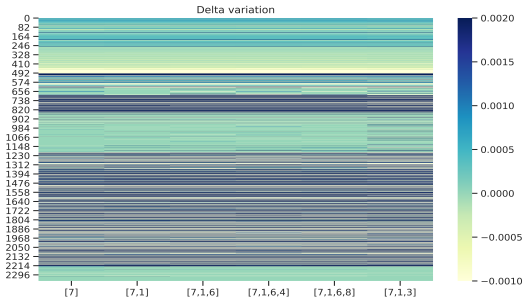

Figure 8: Average of delta variation for malware in the exploration set after being injected with each transformation chain. The first five chains (from left to right) are the best candidates for UAP whereas the less effective chain, (t_7, t_1, t_3), shows visible differences within its features. The Y-Axis represents the mapping of the feature-space for each malware example.

els against such perturbations. In this section, we extend our previous research questions to include:

RQ3. Can we *mitigate* the strength of UAPs against malware classifiers?(§5.2)

RQ4. Can we utilize problem-space knowledge to harden *different types* of classifiers against UAPs? (§5.3)

We introduce our approach to improve defenses, leveraging the UAP attacks in scenarios with limited and perfect knowledge as described in §4. By measuring the effectiveness of new adaptive attacks after the models have been hardened, we determine how each one of the strategies contribute to the robustness of the classifier.

5.1 Adversarial Training with UAPs

A promising mitigation against adversarial examples is adversarial training [32, 48]: introducing evasive examples into the training process which adjusts the decision boundary to cover pockets of adversarial space close to legitimate examples. However, uniformly applying adversarial training to all regions close to the decision boundary can greatly alter the classifier, such that performance suffers on goodware or previously correctly detected malware. Moreover, effort is wasted in securing regions of the feature space which do not intersect with the feasible problem-space region of realizable attacks [58].

Note that in a typical binary classification security detection setting, we only need to protect the model against one type of attacks (e.g., attackers trying to force malware to be misclassified as goodware). Thus, only one class needs to be 'protected' against adversarial examples (compared to multi-class image classification, where each and every class needs to be 'protected'). This means that for adversarially training ML malware detection models, we only need to craft adversarial examples for the malicious class. However, a peculiarity of doing adversarial training considering only adversarial malware is that, when using the standard settings, i.e., computing adversarial counterparts for all the malware examples during training, the TPR of the classifier can be affected significantly. This is because the model learns to accurately detect adversarial malware but not genuine malware, as the model did not see any genuine malware examples during training at all. To avoid this issue, during training we interleave genuine and adversarial malware with a 50/50 ratio. We refer to this as *mixed* adversarial training. In our experiments we also compare against the standard adversarial training, where all the malware examples are adversarial. We refer to this as *pure* adversarial training.

While different ML algorithms necessitate specific adjustments, our process can broadly be defined as follows. *i) Generate problem-space UAPs* using a greedy search on the exploration set to calculate the strongest transformation chain, using the toolkit of transformations available, to quantify the model's intial robustness. *ii) Adversarially train the model*, either by directly introducing newly generated UAPs to the training process (§5.2) or by using synthetic examples derived from the statistical distribution of examples in the first step (§5.3). *iii) Evaluate the robust models* considering an *adaptive attacker*, by performing a fresh search for UAP attacks. We focus on the effectiveness of the UAP attack in terms of UER, and the performance loss incurred on clean data in terms of AUC-ROC and TPR at a fixed FPR of 1%.

5.2 Hardening DREBIN against UAPs

To harden DREBIN against problem-space UAPs and answer **RQ3**, here we instantiate our adversarial training-based defense on the Android malware classifiers. We hypothesize that the linear DREBIN model will not be receptive to adversarial training, as the linear hyperplane will not be flexible enough to adapt to the adversarial inputs, i.e., it will begin to 'forget' patterns of adversarial inputs seen earlier in the training process [42]. To test this, we apply our defense to both the linear model from §4.3, and the non-linear model originally described in §3 which we refer to here as DREBIN-DNN. Both models are implemented using PyTorch [55].

We perform the following steps during each of the the last N epochs of the training procedure. At the start of each minibatch, we apply our attack procedure to the partially trained model and search for the most effective UAP transformation chain, i.e., the UAP that maximizes UER across all true positive examples in the minibatch. Next, this UAP is applied to 50% of the minibatch malware examples—retaining 50% of the clean examples in order to avoid overfitting to the adversarial inputs.

Results. We repeat the PK attack from §4.3 against the non-linear model to act as a baseline (Table 2). For completeness, we also run the LK attack against DREBIN-DNN (Appendix A)—interestingly this model seems to be slightly more

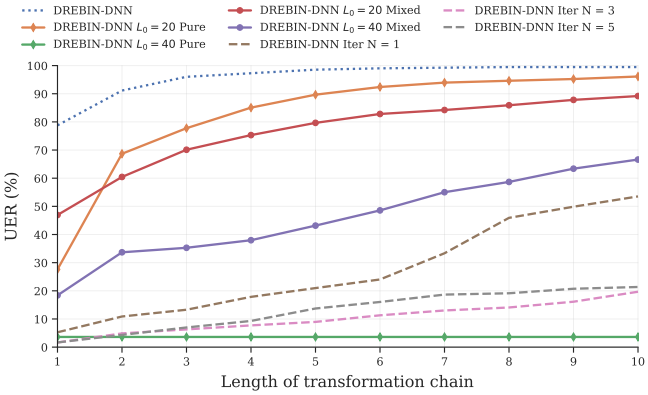

Figure 9: Adaptive attacks against DREBIN-DNN classifiers showing increasing Universal Evasion Rates (UER) at each stage of the transformation chain. Corresponding performance on clean data is shown in Table 2.

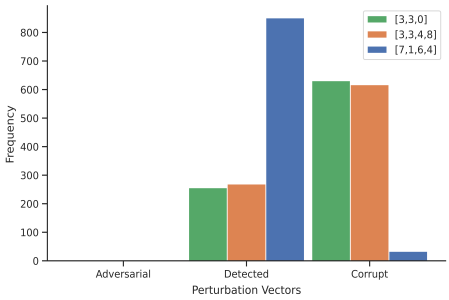

Figure 10: Adversarial examples from previously identified and new adaptive UAP attacks are both unsuccessful in the hardened model either by detection or by forcing the binary to mutate until corruption. Both (t_3, t_3, t_0) and (t_3, t_3, t_4, t_8), are properly detected, making UER close to zero. The remaining transformation chain, (t_7, t_1, t_6, t_4), is the UAP identified against the baseline model and is also correctly detected.

robust to the PK attack, but is generally more vulnerable to the LK attack. We also compare against a number of defenses obtained by generating adversarial examples in the feature space instead of the problem-space. These defenses take two parameters: the L_0 constraint on the perturbation size and the percentage of adversarial examples to include during the adversarial training procedure. For these we consider L_0 constraints of 20 and 40, and adversarial proportions of 50% and 100% ('mixed' and 'pure').

Table 2 shows the results of this procedure applied to the linear DREBIN classifier as well as the non-linear DREBIN-DNN, for the last $N = 1$, 3, and 5 epochs of training, as well as the UER of the adaptive white-box attacks (also depicted in Figure 9). The close results for $N = 3$ and $N = 5$ appear to show an upper bound in the robustness gained, so it is likely that further epochs will result in diminishing returns. The results also confirm our hypothesis that the linear model is not as amenable to adversarial training as the non-linear model. The linear DREBIN model shows a larger performance loss on the clean examples compared to the other models (except for $L_0 = 40$ Pure), and while the robustness is improved for small chains, UER for the chains of length 10 is $> 80\%$.

Overall, the defense that provides the largest improvement in robustness is $L_0 = 40$ Pure, which is the most aggressive feature-space defense we consider, however it comes with a significant performance degradation on non-adversarial examples. $L_0 = 40$ Mixed offers a better trade-off with a fairly large increase in robustness without the performance loss. The other feature-space defenses retain their performance on the non-adversarial examples, but do not show a significant gain in robustness. However, our approach demonstrates an even greater trade-off than $L_0 = 40$ Mixed, with a similarly small performance loss on clean data, but far greater gains in robustness, reducing the maximum UER of length 10 chains from 99.5% to ~20%.

5.3 Hardening EMBER against UAPs

As an answer to **RQ4**, we want to explore how the concept of introducing problem-space information to adversarial training can be adapted and extended beyond neural networks and applied to different machine learning models. Hence, we have adjusted the process to make state-of-the-art classifiers in the Windows domain more resilient to such attacks.

For Windows PE binaries, GBDT classifiers have proved to be highly accurate for malware classification in this domain [6, 7]. However, generating adversarial examples in the problem space is significantly more expensive than in the feature space for Windows than it is for Android. Additionally, while the non-linearity of the GBDT should be receptive to adversarial training, the model is not trained in batches across multiple epochs as is the DNN.

To overcome this limitation, we generate an approximation of the feature-space perturbations induced by the problem-space toolkit. For this we create a statistical model where, for each feature, we compute the probability of the feature being modified as a result of the problem-space UAP attack. At training time, we generate adversarial malware in the feature space by sampling random perturbations using this statistical model. This allows us to significantly reduce the number of problem-space adversarial objects that need to be generated. Note that our approximation does not take into account possible interactions between features in the feature space, i.e., the statistical model assumes independence across features, and although this is likely not always the case, our empirical results show that even this simple statistical model allows us to harden the ML models against adaptive UAP attacks.

Results. We perform adversarial training using both strategies: pure and mixed. As expected, the former model incurs a heavy cost in clean detection performance, with AUC-ROC of .624, while the latter retains better performance at .797.

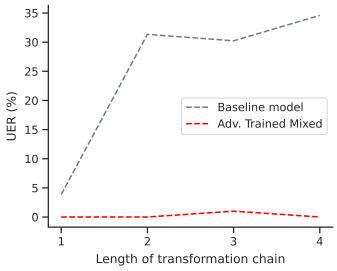

Figure 11: Universal Evasion Rates (UER) between baseline model and models trained with mixed datasets (including clean and adversarial malware examples). After hardening, the best UAP transformation chains are not able to bypass the model.

As such, we continue with the mixed model only.

Following best practices, we perform a fresh adaptive attack against the hardened model. The most effective transformation chains found, (t_3, t_3, t_0) and (t_3, t_3, t_4, t_8), are still successfully detected by the model as shown in Figure 10 and do not represent meaningful threats. Furthermore, we observe in Figure 11 that the model becomes much better at identifying false negatives, as shown by the decrease in UER compared to the baseline. In fact, 99.8% of the attempts are successfully detected. Despite the capacity of the new model to successfully detect most adversarial examples, compared to the DNN models used for DREBIN, in this case the GBDT offers less flexibility to adapt the UAPs during training, which has a negative impact on the detection of genuine malware, i.e., the TPR evaluated on genuine malware decreases.

6 Discussion

Modeling the attacker's constraints. Unlike in computer vision applications, the generation of adversarial examples in the malware domain is subject to specific problem-space constraints that limit how input objects can be modified, to generate realistic working software that preserves malicious functionality [e.g., 26, 45]. Hence, analyzing feature-space robustness of ML models to certain adversarial examples provides an unrealistic view of the models' vulnerabilities. On one hand, many of the attacks available in the feature space are possibly infeasible in the problem space. On the other hand, it can be difficult to model appropriate attacker constraints in the feature space. For example, EMBER contains a mix of discrete and continuous features, making it difficult to model comprehensive feature-space constraints (e.g., using a combination of L_p-norms), even when ignoring some of the problem-space constraints.

UAP attacks. Our experimental results in §4 show that UAP attacks represent an important and practical threat against ML-based malware detectors. However, our results report a disparity on the effectiveness of the attacks for Windows and Android malware. For DREBIN we can craft very successful UAP attacks that, in some cases achieve 100% UER. In contrast, for EMBER the UER of the attacks is approximately 30%. The reason is that for Windows malware we start with a more limited set of transformations to manipulate the malware and, at the same time, the application of these transformations produces non-functional malware in some cases, limiting the capacity of the adversary to create problem-space adversarial objects. In contrast, the set of Android code gadgets that can be used to generate adversarial malware is larger, and the addition of these gadgets does not have as significant an impact on malware functionality.

Defenses. We show that our methodology for adversarially training the models allows us to harden the model against UAP attacks generated with the considered transformation set. However, we cannot guarantee robustness against other possible transformations that could become available for the attackers, i.e., we cannot guarantee robustness against *unknown unknowns*. Compared to adversarial training in the feature space, our methodology focuses on "patching" those UAP vulnerabilities that are more relevant from a practical perspective, without having a significant negative impact on the detection of *clean* malware, in particular for DNNs. Although in our work we consider some LK attacks, we did not consider *transfer attacks*, which can provide a more comprehensive view on the robustness of these models. Additionally, it would be interesting to analyse if the application of this or a similar methodology can be appropriate to mitigate input-specific attacks. These last two points are left as future work.

7 Related Work

Adversarial Examples for Malware. The vulnerabilities of machine learning systems to different threats, both at training and test time, have been investigated for almost 15 years [15, 16, 36], attracting a higher attention in the research community since Szegedy et al. [66] and Biggio et al. [17] showed the existence and weakness of machine learning algorithms to adversarial examples. Although the literature in adversarial machine learning has put the focus on computer vision applications, the security community has also started to evaluate the problem on different malware variants, including (but not limited to) Android malware [27, 34, 58, 76], Windows malware [26, 43, 44, 45, 59], PDFs [17, 65, 75], NIDS [8, 9, 24], and malicious Javascript [29]. It is important to observe that one peculiarity of the malware domain is that *feature mapping* functions are generally not invertible and not differentiable. This implies that translating an adversarial feature vector in the *feature space* to an actual malware in the *problem space* is significantly more complex. To support this challenge, Pierazzi et al. [58] propose a general framework for problem-space attacks which also clarifies which

Table 2: Comparison of our problem-space defenses against a set of feature-space defenses and undefended models, showing performance on clean examples (AUC-ROC, TPR) and robustness against an adaptive attacker (UER at $|\mathbf{T}|$ of 1, 4, and 10).

	MODEL	AUC-ROC	TPR at 1% FPR	UER_1	UER_4	UER_{10}
Undefended	DREBIN	0.981	0.855	98.7%	100%	100%
	DREBIN-DNN	0.992	0.900	78.8%	97.3%	99.5%
Feature-space defenses	DREBIN-DNN $L_0 = 20$ Pure	0.989	0.843	27.6%	85.1%	96.1%
	DREBIN-DNN $L_0 = 40$ Pure	0.903	0.347	3.6%	3.6%	3.6%
	DREBIN-DNN $L_0 = 20$ Mixed	0.990	0.872	46.9%	75.3%	89.2%
	DREBIN-DNN $L_0 = 40$ Mixed	0.990	0.877	18.4%	38.0%	66.6%
Problem-space defenses	DREBIN Iter $N = 1$	0.978	0.775	23.0%	70.4%	95.7%
	DREBIN Iter $N = 3$	0.978	0.766	21.0%	47.0%	87.0%
	DREBIN Iter $N = 5$	0.978	0.761	17.4%	35.1%	82.6%
	DREBIN-DNN Iter $N = 1$	0.990	0.874	5.3%	17.9%	53.5%
	DREBIN-DNN Iter $N = 3$	0.990	0.871	1.6%	7.7%	19.7%
	DREBIN-DNN Iter $N = 5$	0.990	0.872	1.7%	9.3%	20.4%
Undefended	EMBER LightGBM	0.999	0.982	3.8%	34.6%	—
Problem-space defense	EMBER LightGBM w/ Adv Tr	0.988	0.797	0.0%	0.1%	—

constraints need to be defined when considering attacks that handle problem-space objects. In our study, we are interested in studying problem-space attacks, and refer to this framework to define our threat model and constraints.

Universal Adversarial Perturbations. Moosavi-Dezfooli et al. [51] showed the existence of UAPs, where a single adversarial perturbation applied over a large set of inputs can cause the target model to misclassify a large fraction of those inputs. UAPs expose the systemic vulnerabilities of the model that can be exploited regardless of the input [23, 37]. UAP attacks are the basis of many practical and physically realizable attacks across different application domains, such as image classification [13, 19, 23, 40, 52], object detection [28, 47, 64], perceptual ad-blocking [69], LiDAR-based object detection [20, 70], NLP tasks [72], and audio or speech classification [1, 54]. However, to the best of our knowledge, UAP attacks have not been explored in the context of ML-based malware detection.

Different defenses have been proposed to mitigate UAP attacks, most of them only explored in the context of computer vision applications. Some of these defenses aim to detect UAP attacks by trying to denoise the inputs [3, 18] or, in the case of SentiNet [22] aiming to detect adversarial patches in image classification. Other sets of defense aim to harden the model by performing universal adversarial training [53, 62].

8 Availability

We release upon request our UAP attacks and defenses for malware as a library, GAME-UP, to foster future research in the community.

9 Conclusion

After demonstrating that UAPs and input-specific attacks have similar effectiveness in the feature space, we systematically generate and evaluate problem-space adversarial malware using UAPs in both perfect and limited knowledge settings. We build on the results to propose a defense: a new variant of adversarial training, also highlighting that non-linear models such as DNNs are more appropriate than linear classifiers as robust models against problem-space malware UAP attacks.

Acknowledgments

This research has been supported, in parts, by EC H2020 Project CONCORDIA GA 830927.

References

[1] S. Abdoli, L. G. Hafemann, J. Rony, I. B. Ayed, P. Cardinal, and A. L. Koerich. Universal adversarial audio perturbations. *CoRR*, abs/1908.03173, 2019.

[2] H. Aghakhani, F. Gritti, F. Mecca, M. Lindorfer, S. Ortolani, D. Balzarotti, G. Vigna, and C. Kruegel. When malware is packin' heat; limits of machine learning classifiers based on static analysis features. In *NDSS*. The Internet Society, 2020.

[3] N. Akhtar, J. Liu, and A. Mian. Defense against Universal Adversarial Perturbations. In *CVPR*, pages 3389–3398, 2018.

[4] A. Al-Dujaili, A. Huang, E. Hemberg, and U.-M. O'Reilly. Adversarial deep learning for robust detection

of binary encoded malware. In *2018 IEEE Security and Privacy Workshops (SPW)*, pages 76–82. IEEE, 2018.

[5] K. Allix, T. F. Bissyandé, J. Klein, and Y. Le Traon. Androzoo: Collecting millions of android apps for the research community. In *Proceedings of the 13th International Conference on Mining Software Repositories*, MSR '16. ACM, 2016.

[6] H. S. Anderson and P. Roth. Ember: an open dataset for training static pe malware machine learning models. *arXiv preprint arXiv:1804.04637*, 2018.

[7] H. S. Anderson, A. Kharkar, B. Filar, D. Evans, and P. Roth. Learning to evade static pe machine learning malware models via reinforcement learning. *arXiv preprint arXiv:1801.08917*, 2018.

[8] G. Apruzzese and M. Colajanni. Evading botnet detectors based on flows and random forest with adversarial samples. In *2018 IEEE 17th International Symposium on Network Computing and Applications (NCA)*, pages 1–8. IEEE, 2018.

[9] G. Apruzzese, M. Andreolini, M. Marchetti, A. Venturi, and M. Colajanni. Deep reinforcement adversarial learning against botnet evasion attacks. *IEEE Transactions on Network and Service Management*, 17(4):1975–1987, 2020.

[10] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, and K. Rieck. DREBIN: effective and explainable detection of android malware in your pocket. In *NDSS*. The Internet Society, 2014.

[11] D. Arp, E. Quiring, F. Pendlebury, A. Warnecke, F. Pierazzi, C. Wressnegger, L. Cavallaro, and K. Rieck. Dos and don'ts of machine learning in computer security. *CoRR*, abs/2010.09470, 2020. URL http://arxiv.org/abs/2010.09470.

[12] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. L. Traon, D. Octeau, and P. D. McDaniel. Flowdroid: precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. In *PLDI*, pages 259–269. ACM, 2014.

[13] A. Athalye, L. Engstrom, A. Ilyas, and K. Kwok. Synthesizing Robust Adversarial Examples. In *International Conference on Machine Learning*, pages 284–293, 2018.

[14] E. T. Barr, M. Harman, Y. Jia, A. Marginean, and J. Petke. Automated software transplantation. In *ISSTA*, pages 257–269. ACM, 2015.

[15] M. Barreno, B. Nelson, R. Sears, A. D. Joseph, and J. D. Tygar. Can Machine Learning be Secure? In *Procs. of the Symposium on Information, Computer and Communications Security*, pages 16–25, 2006.

[16] B. Biggio and F. Roli. Wild patterns: Ten years after the rise of adversarial machine learning. *Pattern Recognition*, 84:317–331, 2018.

[17] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Šrndić, P. Laskov, G. Giacinto, and F. Roli. Evasion attacks against machine learning at test time. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 387–402. Springer, 2013.

[18] T. Borkar, F. Heide, and L. Karam. Defending against universal attacks through selective feature regeneration. In *CVPR*, pages 709–719, 2020.

[19] T. B. Brown, D. Mané, A. Roy, M. Abadi, and J. Gilmer. Adversarial Patch. *arXiv preprint arXiv:1712.09665*, 2017.

[20] Y. Cao, C. Xiao, D. Yang, J. Fang, R. Yang, M. Liu, and B. Li. Adversarial Objects against LiDAR-based Autonomous Driving Systems. *arXiv preprint arXiv:1907.05418*, 2019.

[21] N. Carlini, A. Athalye, N. Papernot, W. Brendel, J. Rauber, D. Tsipras, I. J. Goodfellow, A. Madry, and A. Kurakin. On evaluating adversarial robustness. *CoRR*, abs/1902.06705, 2019.

[22] E. Chou, F. Tramèr, and G. Pellegrino. Sentinet: Detecting Localized Universal Attacks against Deep Learning Systems. In *2020 IEEE Security and Privacy Workshops (SPW)*, pages 48–54. IEEE, 2020.

[23] K. T. Co, L. Muñoz-González, S. de Maupeou, and E. C. Lupu. Procedural noise adversarial examples for black-box attacks on deep convolutional networks. In *CCS*, pages 275–289. ACM, 2019.

[24] I. Corona, G. Giacinto, and F. Roli. Adversarial attacks against intrusion detection systems: Taxonomy, solutions and open issues. *Information Sciences*, 239: 201–225, 2013.

[25] Corvus Forensics. Virusshare. https://virusshare.com/, 2011. (last visited Jan. 22, 2021).

[26] L. Demetrio, S. E. Coull, B. Biggio, G. Lagorio, A. Armando, and F. Roli. Adversarial exemples: A survey and experimental evaluation of practical attacks on machine learning for windows malware detection. *arXiv preprint arXiv:2008.07125*, 2020.

[27] A. Demontis, M. Melis, B. Biggio, D. Maiorca, D. Arp, K. Rieck, I. Corona, G. Giacinto, and F. Roli. Yes, machine learning can be more secure! a case study on

android malware detection. *IEEE Transactions on Dependable and Secure Computing*, 16(4):711–724, 2017.

[28] K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, C. Xiao, A. Prakash, T. Kohno, and D. Song. Robust Physical-World Attacks on Deep Learning Visual Classification. In *CVPR*, pages 1625–1634, 2018.

[29] A. Fass, M. Backes, and B. Stock. Hidenoseek: Camouflaging malicious javascript in benign asts. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 1899–1913, 2019.

[30] A. P. Felt, M. Finifter, E. Chin, S. Hanna, and D. Wagner. A survey of mobile malware in the wild. In *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, pages 3–14, 2011.

[31] M. Fossi, E. Johnson, D. Turner, T. Mack, J. Blackbird, D. McKinney, M. K. Low, T. Adams, M. P. Laucht, and J. Gough. Symantec report on the underground economy. *Symantec Corporation*, 51, 2008.

[32] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and Harnessing Adversarial Examples. In *ICLR*, 2015.

[33] Google Inc. Virustotal. https://www.virustotal.com/, 2004. (last visited Jan. 22, 2021).

[34] K. Grosse, N. Papernot, P. Manoharan, M. Backes, and P. D. McDaniel. Adversarial examples for malware detection. In *ESORICS*, Lecture Notes in Computer Science, 2017.

[35] W. Hu and Y. Tan. Generating adversarial malware examples for black-box attacks based on gan. *arXiv preprint arXiv:1702.05983*, 2017.

[36] L. Huang, A. D. Joseph, B. Nelson, B. I. Rubinstein, and J. D. Tygar. Adversarial Machine Learning. In *Procs. of the Workshop on Security and artificial intelligence*, pages 43–58, 2011.

[37] A. Ilyas, S. Santurkar, D. Tsipras, L. Engstrom, B. Tran, and A. Madry. Adversarial examples are not bugs, they are features. In *Advances in Neural Information Processing Systems*, pages 125–136, 2019.

[38] Kaspersky Lab. Adwind malware-as-a-service hits more than 400,000 users globally. https://www.kaspersky.co.uk/blog/adwind-rat/6731/, 2021. (last visited Jan. 22, 2021).

[39] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu. Lightgbm: A highly efficient gradient boosting decision tree. In *Advances in neural information processing systems*, pages 3146–3154, 2017.

[40] V. Khrulkov and I. V. Oseledets. Art of singular vectors and universal adversarial perturbations. In *CVPR*, pages 8562–8570. IEEE Computer Society, 2018.

[41] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *ICLR (Poster)*, 2015.

[42] J. Kirkpatrick, R. Pascanu, N. C. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, D. Hassabis, C. Clopath, D. Kumaran, and R. Hadsell. Overcoming catastrophic forgetting in neural networks. *CoRR*, abs/1612.00796, 2016.

[43] B. Kolosnjaji, A. Demontis, B. Biggio, D. Maiorca, G. Giacinto, C. Eckert, and F. Roli. Adversarial malware binaries: Evading deep learning for malware detection in executables. In *EUSIPCO*, pages 533–537. IEEE, 2018.

[44] R. Labaca-Castro, C. Schmitt, and G. Dreo. Aimed: Evolving malware with genetic programming to evade detection. In *2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, pages 240–247. IEEE, 2019.

[45] R. Labaca-Castro, C. Schmitt, and G. D. Rodosek. Armed: How automatic malware modifications can evade static detection? In *2019 5th International Conference on Information Management (ICIM)*, pages 20–27. IEEE, 2019.

[46] Lastline, Inc. Malware-as-a-service: The 9-to-5 of organized cybercrime. https://www.lastline.com/blog/malware-as-a-service-the-9-to-5-of-organized-cybercrime/, 2021. (last visited Jan. 22, 2021).

[47] X. Liu, H. Yang, Z. Liu, L. Song, H. Li, and Y. Chen. Dpatch: An Adversarial Patch Attack on Object Detectors. *arXiv preprint arXiv:1806.02299*, 2018.

[48] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

[49] B. Miller, A. Kantchelian, M. C. Tschantz, S. Afroz, R. Bachwani, R. Faizullabhoy, L. Huang, V. Shankar, T. Wu, G. Yiu, A. D. Joseph, and J. D. Tygar. Reviewer integration and performance measurement for malware detection. In *DIMVA*, volume 9721 of *Lecture Notes in Computer Science*, pages 122–141. Springer, 2016.

[50] S. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard. Universal adversarial perturbations. In *CVPR*, pages 86–94. IEEE Computer Society, 2017.

[51] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard. Universal adversarial perturbations. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1765–1773, 2017.

[52] K. R. Mopuri, U. Ojha, U. Garg, and R. V. Babu. NAG: network for adversary generation. In *CVPR*, pages 742–751. IEEE Computer Society, 2018.

[53] C. K. Mummadi, T. Brox, and J. H. Metzen. Defending against Universal Perturbations with Shared Adversarial Training. In *International Conference on Computer Vision*, pages 4928–4937, 2019.

[54] P. Neekhara, S. Hussain, P. Pandey, S. Dubnov, J. J. McAuley, and F. Koushanfar. Universal adversarial perturbations for speech recognition systems. In *INTERSPEECH*, pages 481–485. ISCA, 2019.

[55] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*, pages 8024–8035, 2019.

[56] J. Patrick-Evans, L. Cavallaro, and J. Kinder. Probabilistic naming of functions in stripped binaries. In *Annual Computer Security Applications Conference (ACSAC)*, 2020.

[57] F. Pendlebury, F. Pierazzi, R. Jordaney, J. Kinder, and L. Cavallaro. TESSERACT: eliminating experimental bias in malware classification across space and time. In *USENIX Security Symposium*, pages 729–746. USENIX Association, 2019.

[58] F. Pierazzi, F. Pendlebury, J. Cortellazzi, and L. Cavallaro. Intriguing properties of adversarial ml attacks in the problem space. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 1332–1349. IEEE, 2020.

[59] I. Rosenberg, A. Shabtai, L. Rokach, and Y. Elovici. Generic black-box end-to-end attack against state of the art api call based malware classifiers. In *International Symposium on Research in Attacks, Intrusions, and Defenses*. Springer, 2018.

[60] I. Rosenberg, A. Shabtai, L. Rokach, and Y. Elovici. Generic black-box end-to-end attack against state of the art API call based malware classifiers. In *RAID*, Lecture Notes in Computer Science, 2018.

[61] A. Shafahi, M. Najibi, A. Ghiasi, Z. Xu, J. Dickerson, C. Studer, L. S. Davis, G. Taylor, and T. Goldstein. Adversarial Training for Free! In *NeurIPS*, 2019.

[62] A. Shafahi, M. Najibi, Z. Xu, J. Dickerson, L. S. Davis, and T. Goldstein. Universal Adversarial Training. In *AAAI*, 2020.

[63] Skylight Cyber. Cylance, i kill you! https://skylightcyber.com/2019/07/18/cylance-i-kill-you/, 2019. (last visited Jan. 22, 2021).

[64] D. Song, K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, F. Tramer, A. Prakash, and T. Kohno. Physical adversarial examples for object detectors. In *12th {USENIX} Workshop on Offensive Technologies ({WOOT} 18)*, 2018.

[65] N. Šrndic and P. Laskov. Detection of malicious pdf files based on hierarchical document structure. In *Proceedings of the 20th Annual Network & Distributed System Security Symposium*, pages 1–16. Citeseer, 2013.

[66] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[67] The UPX Team. Upx: the ultimate packer for executables. https://upx.github.io/, 1996. (last visited Jan. 22, 2021).

[68] F. Tramèr and D. Boneh. Adversarial Training and Robustness for Multiple Perturbations. In *NeurIPS*, 2019.

[69] F. Tramèr, P. Dupré, G. Rusak, G. Pellegrino, and D. Boneh. Adversarial: Perceptual Ad Blocking Meets Adversarial Machine Learning. In *Conference on Computer and Communications Security*, pages 2005–2021, 2019.

[70] J. Tu, M. Ren, S. Manivasagam, M. Liang, B. Yang, R. Du, F. Cheng, and R. Urtasun. Physically Realizable Adversarial Examples for LiDAR Object Detection. In *CVPR*, pages 13716–13725, 2020.

[71] R. Vallée-Rai, P. Co, E. Gagnon, L. J. Hendren, P. Lam, and V. Sundaresan. Soot - a java bytecode optimization framework. In *CASCON*, page 13. IBM, 1999.

[72] E. Wallace, S. Feng, N. Kandpal, M. Gardner, and S. Singh. Universal adversarial triggers for attacking and analyzing NLP. In *EMNLP/IJCNLP (1)*, pages 2153–2162. Association for Computational Linguistics, 2019.

[73] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas. Dueling network architectures for deep reinforcement learning. In *ICML*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 1995–2003. JMLR.org, 2016.

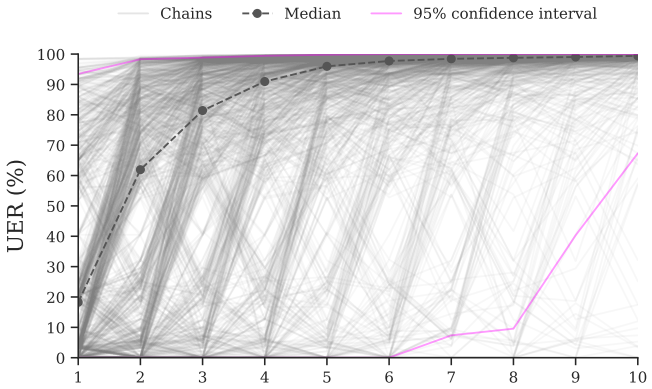

Figure 12: Limited Knowledge (LK) attack against a non-linear DREBIN Android malware classifier.

[74] Webroot Inc. Malware as a service: As easy as it gets. https://www.webroot.com/blog/2016/03/31/malware-service-easy-gets/, 2021. (last visited Jan. 22, 2021).

[75] W. Xu, Y. Qi, and D. Evans. Automatically evading classifiers. In *Proceedings of the 2016 network and distributed systems symposium*, volume 10, 2016.

[76] W. Yang, D. Kong, T. Xie, and C. A. Gunter. Malware detection in adversarial settings: Exploiting feature evolutions and confusions in android apps. In *Proceedings of the 33rd Annual Computer Security Applications Conference*, pages 288–302, 2017.

[77] X. Zhang, A. Mian, R. Gupta, N. Rahnavard, and M. Shah. Cassandra: Detecting trojaned networks from adversarial perturbations. *CoRR*, abs/2007.14433, 2020.

APPENDIX

A Additional Android Result

Figure 12 reports Limited Knowledge (LK) attack against a non-linear DREBIN Android malware classifier, implemented as a DNN with 2 hidden layers. Universal Evasion Rates (UER) are shown for 1,000 random transformation chains up to length 10. Relatively few transformations are required to achieve a high UER, highlighted by the median at each stage. Compared to the linear model (fig. 6), the non-linear model seems even more susceptible to the LK attack, however the flexibility of the model enables us to devise a defense which is better able to adapt to the attacker's set of transformations.