

Evaluating the Robustness of a Production Malware Detection System to Transferable Adversarial Attacks

Milad Nasr^{1†}, Yanick Fratantonio², Luca Invernizzi², Ange Albertini², Loua Farah²,
Alex Petit-Bianco², Andreas Terziser¹, Kurt Thomas², Elie Bursztein², Nicholas Carlini^{1‡}

¹Google DeepMind ²Google

Abstract

As deep learning models become widely deployed as components within larger production systems, their individual shortcomings can create system-level vulnerabilities with real-world impact. This paper studies how *adversarial attacks* targeting an ML component can degrade or bypass an entire production-grade malware detection system, performing a case study analysis of Gmail’s pipeline where file-type identification relies on a ML model.

The malware detection pipeline in use by Gmail contains a machine learning model that *routes* each potential malware sample to a specialized malware classifier to improve accuracy and performance. This model, called Magika, has been open sourced. By designing adversarial examples that fool Magika, we can cause the production malware service to incorrectly route malware to an unsuitable malware detector thereby increasing our chance of evading detection. Specifically, by changing just 13 bytes of a malware sample, we can successfully evade Magika in 90% of cases and thereby allow us to send malware files over Gmail. We then turn our attention to defenses, and develop an approach to mitigate the severity of these types of attacks. For our defended production model, a highly resourced adversary requires 50 bytes to achieve just a 20% attack success rate. We implement this defense, and, thanks to a collaboration with Google engineers, it has already been deployed in production for the Gmail classifier.

1 Introduction

Adversarial examples—evasion attacks that cause machine learning models to misclassify adversarially modified inputs—have been studied for over ten years. Such attacks have been demonstrated on nearly every domain of machine learning, ranging from image classifiers [17, 25, 33] to speech recognition systems [3, 9, 29] to natural language processors [37, 38]. These attacks typically work by performing gradient descent on an input example to maximize its classification loss.

And yet, “real attackers don’t compute gradient gradients” [4]: despite these attacks being known to be possible, there are very few cases where actual adversaries have performed gradient based attacks to cause some specific harm on a production system. Why is this? The most common answer that has been argued for several years [16] is that performing adversarial attacks is rarely the *easiest* method to achieve the adversary’s ultimate objective. For example, adversaries who wish to fool image classification models could simply generate random examples until they find a mistake [16], could perform simple modifications like increasing or decreasing the contrast or brightness, or could make random changes to the

background until one was successful [18]. In this work we show a real world setting where an adversary should use existing adversarial example based approach instead of other modification to misclassify the input.

In particular, we study the question of how an adversary could construct malware that evades detection by the Gmail malware detection system using adversarial approaches [28]. Typically, producing an evasive malware sample requires the laborious process of trying to construct a file that leverages advanced obfuscation techniques, to be both malicious while also hiding its behavior from malware detection. But we find that an alternate approach is possible.

In production environments, malware detection is slow and expensive; it is not practical to run the most advanced malware classifiers over every potential sample. And so, to improve efficiency, detection is often split into three phases. First, a fast signature-based classifier scans for obvious features of general malware. Then, if nothing is detected, a second classifier *routes* this sample to one (or more) specialized malware classifiers. Third, this specialized classifier scans for more sophisticated forms of malware of a particular type (e.g., PDF malware or JavaScript malware). This optimization approach reduces the cost of system significantly lower, however, introduces a critical dependency: the security of the entire system relies on the robustness of each component, including the routing classifier. Google, for example, uses their Magika [15] content-type classifier (in addition to more traditional content-type classifiers) “to help improve Google users’ safety by routing Gmail, Drive, and Safe Browsing files to the proper security and content policy scanners. [8]”

We make the observation that this second classifier introduces the potential for a simple evasion attack: by generating adversarial examples that evade the Magika classifier (in addition to evading the other existing classifiers), it is possible to completely bypass the significantly harder task of fooling the sophisticated malware classifier, by instead causing Magika to route the sample to a classifier that is not specialized to detect it. Because Magika is “just” a standard neural network designed without adversarial examples in mind we can apply standard techniques from the literature to evade detection. Therefore, all that is necessary to produce evasive malware is to (1) evade simple signature detection, and then (2) fool the router to misroute the sample towards the wrong specialized classifier. As we show, this is more easily achieved due to Magika’s open-source codebase. We note that Magika is explicitly never positioned as being robust against adversarial attacks; in fact, the authors call out this limitation and are “looking forward to seeing adversarial examples from the community.”¹ [15] (This paper

[†] Currently at OpenAI

[‡] Currently at Anthropic

¹<https://github.com/google/magika>

does exactly that.) Nonetheless, Gmail decided to add Magika as an additional and complementary file-type classifier to its critical production system, due to Magika’s higher precision and recall on many critical content types. This meant that ultimately Gmail relied on Magika’s robustness for those content types. This is likely a common design pattern in many security systems, which rely on components that are not sufficiently mitigating adversarial attacks.

Contributions. The focus of this paper is in analyzing the Magika classifier as a case study for what can go wrong when machine learning models get integrated into larger systems: the space of the attack on a production system will include attacks on any machine learning component.

Specifically, we show how to leverage recent advances in constructing discrete NLP adversarial examples [38] and apply these attacks to generate evasive malware samples. By changing just a handful of bytes in any given file, we show how to cause the open source Magika router to send common malware file types (e.g., .doc, .pdf) to the incorrect backend malware detectors.

We then show that these adversarial malware samples allow us to fool the production Google classifier. To validate this attack, we work with researchers at Gmail to construct malware samples that we show can evade detection by the entire pipeline.

Next, we turn our attention to defenses. We consider several techniques from the literature and develop our own refinement that we show can significantly increase the difficulty of constructing effective transferable adversarial examples in practical settings, and we then work with Google engineers to deploy this classifier to production to improve users’ safety.

We argue that while the vast literature has shown the difficulty in fully solving the adversarial example problem, a practical security engineering mindset is crucial for deployed systems. The goal should be to implement defenses that significantly increase the cost for the adversary, making the ML component sufficiently robust in its operational context so it is no longer the weakest link, even if theoretical vulnerabilities remain.

Finally, we conclude by discussing the relative costs and benefits of open sourcing machine learning systems used in production environments. While open-sourcing Magika might enable easier attacks, it also allows security researchers to study real-world scenarios and develop more effective defenses.

2 Problem Statement and Background

We begin with a review of malware detection, file type detection, adversarial examples, and attacks on natural language processing (NLP) models.

Malware Detection. Malware includes any malicious program that subverts a user’s device to engage in harmful activities (e.g., steal credit cards and other data, waste system resources to mine cryptocurrencies, send spam, and more). Security systems that detect and prevent malware from executing often rely on a defense-in-depth approach. One layer of this defense is preventing attackers from distributing malicious files via email. Cisco for example reports

blocking 9 million malicious emails per hour², and Google extensively applies malware detection to prevent spread via Gmail [21].

Given the widespread malware threat, it’s no surprise that researchers have devoted considerable effort to developing methods for collecting, studying, and mitigating malicious code [24]. Most relevant for this paper, recent malware detection methods specialize to specific types of malware, enabling an increase in detection accuracy and reduction in processing time—something important when scanning billions of emails per day [8].

File Type Detection. In order to enable filetype-specific malware detection, it is first necessary to detect the type of the file. Simple file type detection tools such as the `file` [11] utility work by scanning the file with basic regular expressions. This works because most file types are designed to be easy to identify in benign scenarios, and so it is only necessary to look at the first few bytes of the header or footer. To evade detection by simple file type detection, prior work [1] has already studied and developed effective techniques, which usually involve finding inconsistencies between the patterns in these systems and how target applications can process the files. For example, in many cases if the adversary adds a different header to the file, the applications might skip the unrecognized header and start from the known patterns while naive file type detection applications that only look at the first few bytes would not detect them.

Magika. To address the shortcomings of existing file type detection approaches, Fratantonio *et al.* [15] trained a machine learning model to detect the type of any given file. This technique, released as an open-source tool called Magika, is used at Google [8]. Specifically, as reported by Google, “Magika significantly boosts Google users’ safety by accurately routing Gmail, Drive, and Safe Browsing files to appropriate security scanners, improving file type identification by 12% (F1 score) compared to previous rule-based systems. This accuracy increase allows for 11% more files to be scanned with specialized malicious AI document scanners, reducing the rate of unidentified files to just 3%.”

The version of Magika discussed in [15] employs a convolutional architecture for file classification. It extracts the first, middle, and last 512 bytes of a file, using their direct byte values as input to the model.³ If a file is smaller than the required size, the input is padded to the appropriate size. After the bytes are processed by a standard deep neural network, the model outputs a probability over expected content types.

While, for some content types, Magika is a strict improvement over the `file` utility in terms of benign accuracy on non-adversarial data, it was also not designed for robustness [15]. The focus of this paper is to (1) investigate the security consequences of deploying Magika in settings with adversaries, and then (2) improve its robustness to attack so that it can be used more reliably.

Adversarial examples. An adversarial example is an input designed to deceive a machine learning model into making an incorrect prediction. Adversarial examples are crafted by adding subtle

²https://www.cisco.com/c/dam/global/en_au/pdfs/talos-ir-quarterly-threats-2024-q2.pdf

³Newer versions of Magika use a slightly different architecture and parameters; such differences are not relevant in the context of this paper.

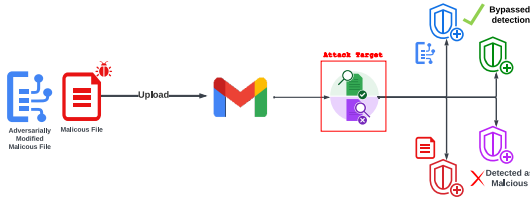


Figure 1: Overview of our attack strategy against Gmail. An attacker aims to send a malicious file as an email attachment. The attacker adversarially modifies the file such that the Magika file-type detector run by Google’s servers misroutes the file to an incorrect, specialized security scanner (e.g., a PDF is sent to a Windows executable scanner), thus evading detection. Absent these adversarial modifications, the file would be sent to the correct security scanner and be accurately detected as malicious.

perturbations to the original data so that the object remains the same (e.g., images look the same, audio sounds the same, text means the same). Mathematically, this can be expressed as the following:

$$\mathbf{x}^* = \mathbf{x} + \arg \min \{z : O(\mathbf{x} + z) \neq O(\mathbf{x})\} = \mathbf{x} + \delta_{\mathbf{x}} \quad (1)$$

where \mathbf{x} is the original input, \mathbf{x}^* is the adversarial perturbation, and $O(\cdot)$ represents the model’s prediction.

There are many techniques to generate adversarial examples [17, 25], depending on the norm being minimized and the domain being attacked. Since the first adversarial attacks were developed, researchers have also attempted to develop defenses that reduce the efficacy of these attacks; unfortunately, none have managed to entirely eliminate the underlying vulnerability to adversarial attack.

Generating natural language adversarial examples. Due to the rapid progress in the field of natural language processing [7, 12, 30], there has been significant interest in developing adversarial examples for language models. Early attacks were able to construct English sentences that fooled sentiment analysis systems [14] by simply adding or removing whitespace, or inserting and removing a few characters [20, 22]. Subsequent attacks fooled summarization and question answering models by inserting unrelated sentences [26]. But recent language models are significantly more advanced, and thus robust to these simple attacks [31, 38].

As a result of the fragility of early attacks, recent work has introduced stronger techniques that more reliably fool NLP systems. We build our attack on the techniques of Zou *et al.* [38], who introduce the Greedy Coordinate Gradient (GCG) attack to generate adversarial examples over discrete input domains with limits on the number of changes made to the input similar to our setting. This attack was initially designed to introduce a new *adversarial suffix* to the end of a given user command to a language model, but we will show that it is not too hard to adapt this attack to our setting where we in-place modify particular bytes of a file.

Robustness to Transferable Adversarial Examples. Defending against adversarial examples is a complex challenge, particularly due to

their transferability between unrelated models, as demonstrated by Liu *et al.* [23]. This transferability makes it difficult to definitively evaluate the robustness of defenses. While achieving complete immunity remains an open problem, several defense strategies have been explored. Tramer *et al.* [35] employed ensemble methods, combining predictions from multiple models to mitigate the impact of transferable adversarial examples. This work focuses on evaluating adversarial training, demonstrating its significant impact on model accuracy. Similar to what we will do, Shumailov *et al.* [32], develop approaches to prevent transfer attacks with a cryptography-based approach.

3 Attack Objective and Threat Model

Our attack objective is straightforward: we aim to send malware via an attachment in Gmail as shown in Figure 1. As discussed, Gmail’s malware detection consists of lightweight signature matching, followed by one or more file-type routers, one of which is called Magika, that sends samples to the most appropriate anti-virus scanner (e.g., PDF scanner, Windows executable scanner). Our attack objective focuses on the routing logic.⁴

We assume the adversary initially constructs the malware sample without accounting for any possible antivirus defenses, yielding some document or executable that will perform the desired behavior. Then, the adversary takes this file and attempts to modify it so that it will successfully evade detection by a given set of security classifiers. Splitting the attack into two steps in this way does, in theory, make the attack “harder”: conceivably one could design the malware with knowledge of the detection techniques and thus make the evasion easier. But in practice, the expertise needed to develop effective malware is often different from that of constructing adversarial examples, and therefore this dividing of responsibilities is, we believe, more realistic. We consider two possible settings of adversary knowledge to construct the evasion attack.

Full Whitebox. The first threat model we consider is a *full white-box* threat model, where an adversary has complete access to the model including the exact parameters used in deployed model. While we will show that attacks under this threat model are extremely effective, they require the (strong) assumption that an adversary for some reason has access to the model parameters. Despite this, in Section 4 we show that this is a realistic threat model for attacking Magika.

Black-box. The second threat model considered in this paper is that of the black-box adversary who still has the same attack objective, but where the defender is allowed to keep a relatively small set of secrets that defend the system. This is a common practice in many security applications (e.g. secret-key cryptography, address space layout randomization [34], stack canaries, or pointer authentication).

In Section 5, we evaluate several techniques to mitigate Magika’s vulnerability to attack under this threat model. Limiting the adversaries knowledge in this way does not completely prevent all forms of attack. Specifically, two general attack strategies remain: (1) *query attacks* [5, 10] make oracle queries to the deployed system

⁴We assume that signature matching is bypassable by an attacker who uses a new malicious file not present in a previous attack campaign.

in order to understand the way the deployed classifier behaves on specific inputs, and adapts the attack accordingly; and (2) *transfer attacks* [27] generate white-box adversarial examples on one classifier, and replay them on another classifier in the hope they will remain effective.

3.1 Distortion metrics

Throughout this paper we evaluate our attacks based primarily on whether or not they succeed. But because we can eventually achieve 100% success rate in every instance for undefended models we additionally break down the success rate of our attack as a function of the number of bytes changed in the malware sample.

In some machine learning settings, measuring distortion as “number of changes introduced” (e.g., number of pixels perturbed for an image classifier, or number of words changed for a natural language processing model) makes sense. There are two reasons for this:

- First, an image or natural language sentence is only an adversarial example if it would still be labeled by a human as the original label; thus, the human acts as the oracle who decides the label of the image or sentence.
- Second, most images or text are shown to humans and machines simultaneously, and so introducing extreme noise would, if not be detected as malicious by the machine, be detected as malicious by the human.

But in the space of malware, neither of these motivations are relevant as demonstrated by polymorphic strains of malware where attackers produce arbitrarily diverse files via unpacking, encryption, or functionally-equivalent code snippets. Humans are not the judge of whether or not a file is malicious or not; it either performs malicious behavior or it doesn’t. If, having modified 1000 bytes, the sample is still malicious, then it is still a valid malware sample. And further, there is no inherent reason why attacks that change 100 bytes of a file are any “more obvious” than attacks that only changed 10 bytes—modifying the bytes within an HTML comment, no matter how many, does not alter the visual appearance of the rendered web page.

For both of these reasons, measuring attack success rate as a function of the number of bytes perturbed is not necessarily an optimal metric. But we nevertheless believe it is still useful. For one, measuring the number of bytes perturbed allows us to quantify the “difficulty” of the attack—as has been shown in past research on adversarial malware, it is easier to evade a malware classifier by changing a larger fraction of the bytes in a file than a smaller fraction. And so while it is not important that we make this number small, the fact that it can be small indicates the inherent vulnerability of these models. Additionally, measuring the number of bytes helps to compare defense approaches.

4 Attacking Magika

We now turn our attention to the technical question of how we can attack Magika to make it mistake files of one type for another type. Once this is achieved, Magika will then route the malicious sample to the wrong specialized malware detector, thus degrading detection. We focus on evading Magika as the measure of success in most of the work. We will also demonstrate a proof of concept that showcases the evasion of the entire malware detection system.

Algorithm 1 Greedy Coordinate Gradient

Require: Initial prompt $x_{1:n}$, modifiable subset \mathcal{I} , iterations T , loss \mathcal{L} , k , batch size B
loop T times
 for $i \in \mathcal{I}$ **do**
 $\mathcal{X}_i := \text{Top-}k(-\nabla_{e_{x_i}} \mathcal{L}(x_{1:n}))$ \triangleright Compute top- k promising token substitutions
 end for
 for $b = 1, \dots, B$ **do**
 $\tilde{x}_{1:n}^{(b)} := x_{1:n}$ \triangleright Initialize element of batch
 $\tilde{x}_i^{(b)} := \text{Uniform}(\mathcal{X}_i)$, where $i = \text{Uniform}(\mathcal{I})$ \triangleright Select random replacement token
 end for
 $x_{1:n} := \tilde{x}_{1:n}^{(b^*)}$, where $b^* = \arg \min_b \mathcal{L}(\tilde{x}_{1:n}^{(b)})$ \triangleright Compute best replacement
end loop

We hypothesize that Google used the same version as the open sourced implementation in their product. We were able to confirm this by sending several samples through Gmail and observing the behavior is similar to the open sourced version. (As we will discuss in the ethics section, we obtain advance permission from Gmail and ensure no malware is transmitted to any other user than ourselves.) Because the open source implementation of Magika shares the exact same set of parameters as the production version of this classifier used in Gmail (at the time of disclosing the attack), we can implement an entirely white-box attack. Please note that no real users were ever targeted in this work, we will provide additional details in the ethical section of the paper.⁵ To a large degree, our attack here is a straightforward application of GCG [38], but with one key change: instead of searching for a suffix that can be appended to a paragraph to change the classification, we search for which bytes can be in-place modified to change the classification. The bytes we modify are so-called *blind spots* [6]: bytes of a file that can be modified without disturbing the structure of the file and still can be parsed as the correct format and maintain the malicious functionality. The key metric in this attack that measures the attack difficulty is the number of bytes modified, and not the specific values they are changed to. Hence, our focus is primarily on l_0 attacks that aim to minimize the number of bytes modified. Algorithm 1 summarizes our attack for attacking Magika in white-box setting. The schema of the attack is also provided in Figure 2.

4.1 Experimental Setup

Dataset. We evaluate our attacks and defenses on 1,130 files, sampled by selecting 10 random test-set data points from each of the Magika’s 113 file types supported at the time of writing. The dataset was provided directly to us by the authors of Magika [15]. This same set of 1,130 files is used across all experiments throughout the paper.

In addition to the evaluation dataset, we followed the data collection methodology described in the Magika paper to gather a dataset

⁵As a result of our experiments, and the defenses we will evaluate in Section 5, the version of Magika used in production is no longer the same as the open-source implementation.

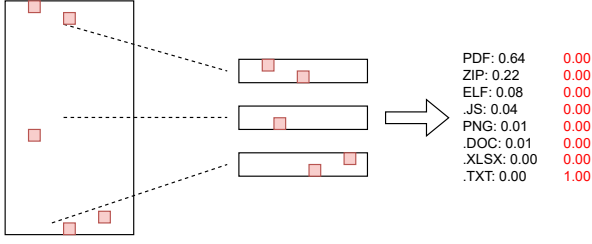


Figure 2: Schematic of our attack. Magika processes an arbitrarily large file by first extracting just 1536 bytes (the first, middle, and last 512 bytes), and then classifying these 1536 bytes with a neural network. By modifying just five of these bytes, we will show how to successfully cause the classifier to assign arbitrary incorrect file type to files, and thus route the malware sample to the wrong specialized classifier.

(20M files) from a similar distribution (VirusTotal [36] and Github). This dataset was used to train our substitute models for evaluation and transferability experiments.

Implementation. We reimplemented Magika in JAX, initializing the model with the provided Keras weights, and verified that our implementation performs equivalently to the original. We also implemented the alternative architectures and various defenses discussed in the following sections in JAX, using data distributions similar to those used in the original evaluation.

For transfer attacks, we followed the standard approach in adversarial machine learning: the attacker trains a substitute model on a dataset drawn from a similar distribution. The substitute models differ from the target Magika model in initialization and data shuffles.

Preprocessing. Unless stated we follow Magika’s default pipeline, each file is represented by a 1,536-byte sequence formed by concatenating three fixed-length slices: the first 512 bytes, a 512-byte segment sampled from the middle, and the last 512 bytes.

Ensemble variant of GCG. In experiments where the attacker has access to multiple substitute models, we adapt the Greedy Coordinate Gradient (GCG) attack (Algorithm 2) to optimize perturbations jointly across all models in the ensemble. Given a set of N substitute classifiers $\mathcal{E} = \{f_{\theta_1}, \dots, f_{\theta_N}\}$, we compute the gradient of the average loss over all models at each iteration. Candidate byte modifications are scored according to this ensemble loss.

4.2 Warm up

To begin, we evaluate how vulnerable Magika is to adversarial examples without considering real-world constraints. For example, many file types have magic bytes that must never be changed; or have checksums that must be valid for the file to correctly process. For the moment, we disregard these real-world constraints in order to develop an understanding of the vulnerability of the classifier in isolation. Then, in the following section, we will re-introduce these “problem space” constraints [28] and show the attack remains effective.

Algorithm 2 Greedy Coordinate Gradient with Blind Spot

Require: Initial bytes $x_{1:n}$, blind spot positions \mathcal{B} , iterations T , loss \mathcal{L} , k , neighbor search size B

loop T times

$\mathcal{X}_i := \text{Top-}k(|\nabla_{e_{x_i}}| \mathcal{L}(x_{1:n}))$ \triangleright Compute top- k promising substitutions

for $b = 1, \dots, B$ **do**

$\hat{x}_{1:n}^{(b)} := x_{1:n}$ \triangleright Initialize element of search batch

$\hat{x}_i^{(b)} := \text{Uniform}((\nabla_{e_{x_i}}[\mathcal{X}_i] < 0) ? [0, x_{1:n}[\mathcal{X}_i]] : (x_{1:n}[\mathcal{X}_i], 255))$ \triangleright Select random replacement token

$x_{1:n} := \hat{x}_{1:n}^{(b^*)}$, where $b^* = \arg \min_b \mathcal{L}(\hat{x}_{1:n}^{(b)})$ \triangleright Compute best replacement

end for

end loop

Ensure: Optimized prompt $x_{1:n}$

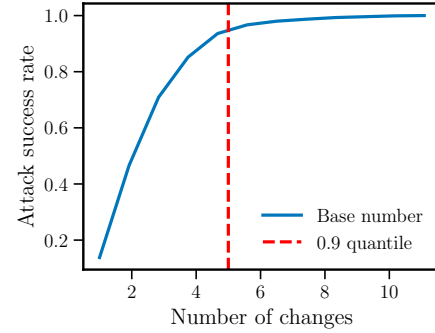


Figure 3: Attack success rate as a function of the number of bytes modified. By modifying just 5 bytes, our attack can cause 90% of malicious files to be misclassified by Magika.

Attack Procedure. The adversary receives a malware sample as input, and has access to the weights of the Magika system in order to compute the gradients of the classification loss with respect to the input bytes. With this access, the attacker runs Algorithm 2: first, identifying which input bytes have the greatest impact on the model’s classification, then, selecting the k positions with the largest gradient values, and finally, modifying each of these positions individually modified with all possible values. The adversary takes each of these candidates, evaluates the classification loss for each modified position and value, and ultimately chooses the modification that results in the lowest loss.

Evaluation. We report the average attack success rate over all of the file type in Figure 3. We find that Magika will misclassify nearly every category of file with just a few edited bytes.

As we can see in Figure 3, it is straightforward for an adversary to cause Magika to misroute any given file type. Over half of binaries can be misclassified with fewer than three bytes modified; and nearly all can with just ten.

A few-byte modification is a nearly infinitesimal modification: many of our files are several megabytes and so a three byte modification represents just 0.00001% of the file. However, the reason this

perturbation is so small is that Magika only inspects a relatively small fraction of the bytes in the file; specifically, recall that Magika samples the first, middle, and last 512 bytes of the file. Therefore, while it is true that the *absolute* magnitude of the perturbation is exceptionally small, the magnitude of the perturbation relative to the input size to the classifier is in line with other adversarial attacks at a ratio between 0.1-1%.

We believe the difference between these two quantities highlights the difference between attacks as measured in the real world and attacks looking only at the machine learning model in isolation. If the classifier is used in such a way that it only inspects a negligible fraction of the file, then an attacker can completely disregard any bytes not processed by the classifier. In later sections, we will discuss defenses including approaches that randomize the bytes which are scanned to improve the robustness of this classifier to attack.

4.3 Format Preserving Attack

We now adjust the prior attack to construct malware that works in the problem space, “wherein the challenge lies in modifying real input-space objects” [28].

There are two types of constraints we must satisfy in order for a file to remain valid:

- *Operating system constraints.* The operating system must believe that the file is of the type intended by an adversary, otherwise when the victim receives the malware and attempts to open it, the operating system will refuse to do so. Each operating system has a different (often exceptionally simple) file-type classifier that, e.g., looks at the first bytes of a file to check the presence of so-called “magic bytes”. For example, PDF files typically must begin with bytes like %PDF-1.5 in order to be valid. A naive file type detector looks at the beginning of the file and if they see %PDF-1.5 they identify them as PDF which then can be used by the operating system to use a PDF reader to open the file. But it is not sufficient for the OS to choose to open the correct program.
- *File-specific constraints.* Nearly all file types have a much more restricted structure that must be valid for the file to actually run. For example, many file types contain metadata information (like the size of buffers or pointers to other data). Other file types expect a given structure (e.g., XML-based file formats would not parse if an adversary broke the XML tags).

An attack that causes Magika to mis-identify one file type as another but breaks either the OS or filetype constraints is invalid. In this section we develop attacks that are effective taking these constraints into consideration. Thus, in contrast to many scenarios where adversarial examples primarily focus on designing attacks that deceive the classifier alone, this section emphasizes on preserving the file format. For instance, in the PDF example above, we now require that any modifications must ensure that the resulting file remains a valid PDF. This means that crafting effective attacks necessitates a thorough understanding of the file format in question.

4.3.1 File types. We evaluate the practicality of our attack on several popular file types that are commonly used to transfer malicious

files, with the understanding that any other file type will likely behave qualitatively similar to these case studies.

To modify file contents while preserving the functionality, we need to find blind spots [6], parts of the file that can be freely modified without altering the functionality of the file. File formats have different characteristics that make them more or less difficult to modify [2]. For this reason, we make use of the Mitra⁶ tool to quickly identify bytes that are able to be modified for 40 different file formats. From a blind spots’ perspective, file formats fall in three categories:

Full Control. Some file formats tolerate to be present at any offset in the first megabytes of the file, allowing for unlimited control right from the beginning: typically, archives such as ZIP, Rar, 7zip, and hardware images such as ISO and Microsoft VHD (if the boot sectors aren’t used, as the file systems are typically defined in further sectors). For these formats, an adversary can typically modify all of the first 2048 bytes of the file.

Parasite. Most file formats (e.g., executables, documents, media format) can tolerate comments or extra metadata.

For example, GIF images have no room in their headers, so instead an adversary can make the header smaller by relocating the global palette to each frame, and abuse the first frame comments instead. Some of these configurations may be less compatible under specific environment, and some checksums may need to be updated (for TAR, PNG or archives), but this will give typically most (at least 2000) controllable bytes of the first 2048 bytes of the file for the following formats.⁷

Scattered Blind Spots. Some formats have no clearly abusible range or relocatable information. In this case, the abusible data only comes from some deprecated fields or descriptive text data, such as sections/segment names, title, or used application. Inserting early null characters in fixed fields to abuse the rest of the field is a common approach.

Overall, a vast majority of the common binary file formats can be abused, and room can be made even in arbitrary files for more than 1000 bytes in the first 2048 bytes while retaining the original file functionality.

Format Preserving Adversarial Examples. In general we create a custom process for each different file type that gets an unmodified file as input and return a version of the file that is functionally equivalent and a list of bytes that can be modified. After having a few free bytes in each file format we now measure the success rate of our attack on this setting. We focus on seven commonly used file formats (PDF, ZIP, Docx, Xlsx, ELF, PNG and Javascript) to have coverage over different approaches and file types used to deploy malware.

Figure 4 summarizes our attack success rate across each of the seven file types. Each of these attacks preserve the original file formats, and their original (malicious) functionality remains valid.

⁶<https://github.com/corkami/mitra>

⁷Other examples of parasite files are executables (PE, Mach-O, ELF, APK, JAR (via ZIP)), documents (HTML, PDF, RTF, DOCX, XSLX, PPTX (via ZIP)), images (GIF, PNG, JPG, JP2 (via MP4), TIFF, Photoshop, DICOM), video (MP4, MKV), audio (MP3 (via ID3v2 header), WAV (via RIFF container)), archives (GZip, TAR).

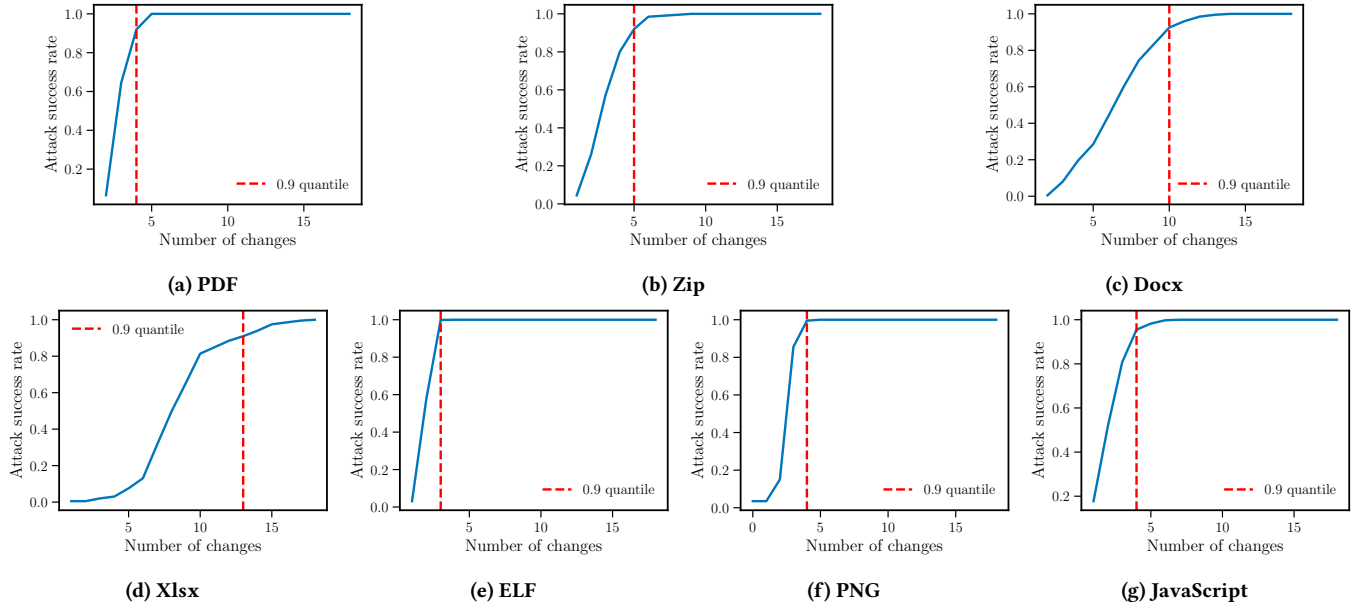


Figure 4: The cumulative number of changes required to the classify each file format file preserving the format.

Specifically, this means we maintain the necessary header for all formats, and modifications are limited to non-critical portions such as code comments in JavaScript and PDF, file names in ZIP files, editing extra jumps in ELF files, Text chunks in PNGs, and comments and names in Office files. Office files required the most modifications compared to other formats. However, all file formats demonstrated a high degree of vulnerability and were easily affected. Despite these challenges we achieve a 100% attack success rate for all file types. If we constrain our edits to just 13 bytes, we can achieve 90% on all file types.

4.4 An end-to-end attack on Gmail

To demonstrate the potential impact of adversarial attacks against Magika, we conduct an end-to-end experiment showing that it is possible to transmit malware via Gmail under certain conditions.

While our study focuses specifically on Magika’s role in Gmail’s attachment scanning pipeline, our observations suggest that Gmail likely employs a defense-in-depth strategy with additional filtering mechanisms beyond Magika. In particular, we observed that rule-based checks on file extensions influence the routing and filtering of attachments. For example, attempting to upload a file named `sample.exe` is immediately blocked by the Gmail user interface, preventing it from ever reaching Magika. Similarly, a malicious PDF that retains its `.pdf` extension appears to be routed to a dedicated PDF malware scanner, bypassing Magika’s classification entirely. To allow our proof-of-concept attack to reach Magika, we removed or altered file extensions, thereby avoiding both hard blocking (as with `.exe`) and forced routing to specialized scanners (as with `.pdf`).

For the attack itself, we selected a malicious PDF sample from VirusTotal [36] that was not detected by popular antivirus scanners (to avoid triggering signature-based detection). We then applied our adversarial perturbation, additionally modifying the file to evade

traditional signature-based file-type detection tools (e.g., `file`), and removed the `.pdf` extension before uploading the attachment to Gmail.

Then, we found bytes of the PDF that could be altered without corrupting the file format, and modified them in such a way that Magika no longer classifies the file as a PDF. When we again sent the modified malicious PDF through Gmail, we observed that the message was successfully delivered to a second account under our control (see Figure 5). Our results demonstrate that by bypassing Magika and other file-type detection techniques, we are able to successfully route our malicious files to an invalid scanner, defeating protections.

One potential limitation with this proof of concept is that after the victim receives our malicious PDF, the lack of a file extension might impact whether their device will still open the file. Fortunately for the attacker, depending on a victim’s operating system, we can design our PDF such that the process that decides what application to open the unknown file with—such as `gnome-open` or `xdg-open`—still operates correctly. This is feasible because the logic used by an operating system differs from the file-type detection used by Gmail. Figure 5 shows a malicious PDF that can successfully evade Gmail’s security scanner, but that will correctly open with the PDF viewer on Ubuntu 22. Alternatively, for other cases, the adversary could attempt to social engineer the victim to add the missing extension back to the file or ask them to execute it directly.

5 Defenses Approaches

Given the vulnerability of Gmail to white-box adversarial example exploitation methods, we now take steps to develop defenses that mitigate the efficacy of this attack. Specifically, we aim to answer an entirely practical question: is it possible to increase the difficulty of exploiting the Magika router component of the defense

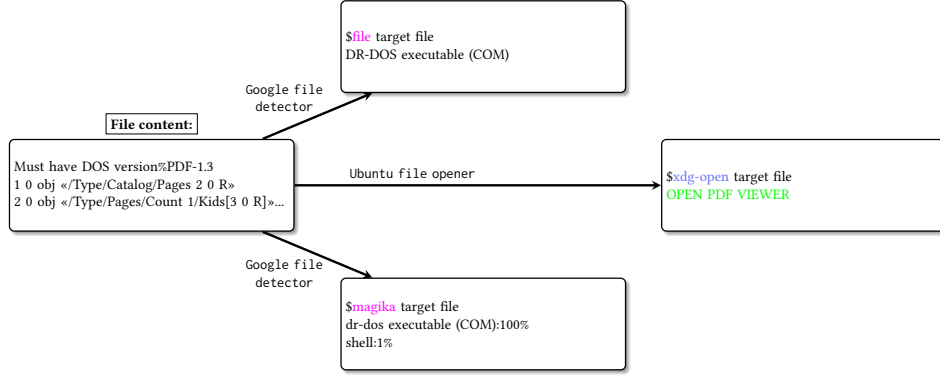


Figure 5: A proof of concept for the attack that can be used in Gmail to launch an end to end attack. Given the different in how different operating system select what applications to open the files

so that it is no longer the weakest defense component? Instead of aiming to completely solve the problem of adversarial examples (something that the research community has been unable to achieve despite over a decade of research), our lower bar is both more practical and more realistic.

Towards that end, in this section we design and evaluate defense strategies that operate under the black-box *transferability* threat model. In collaboration with Gmail developers, we evaluate several strategies where the parameters of the production system are set differently than those of the open-source release. We investigate how various standard defense techniques like randomization would apply to this malware classification setting, but ultimately find none are effective at significantly decreasing the attack success rate. As such, we develop our own defense strategy and show it significantly reduces the vulnerability of the production system to transfer attacks.

Threat Model. We adopt a standard *transfer attack* setting from the adversarial ML literature. The defender deploys a proprietary model $f_{\theta}^{\text{target}}$ trained on internal data and does not release its parameters or weights. The attacker does not have direct access to $f_{\theta}^{\text{target}}$, but can: (1) access the open-source Magika architecture and training code, (2) collect data drawn from a *similar distribution* as the defender’s training data (but not the exact dataset), and (3) train one or more *substitute models* $f_{\theta'}$ using this public information and sufficient compute resources. The attacker crafts adversarial perturbations against $f_{\theta'}$ and attempts to transfer them to $f_{\theta}^{\text{target}}$. For each defense we will describe the approach used to train the transfer models. We also assume the adversary has complete knowledge of the algorithm used in the training except the explicit randomness or private secrets of the training.

5.1 Different Randomization

Perhaps the most straightforward defense idea is simply to release a retrained model instead of the original target model. Because training machine learning models involves a large degree of randomness, this retrained model will have different learned parameters, and attacks crafted on the open-source model may not directly transfer to the closed-source model.

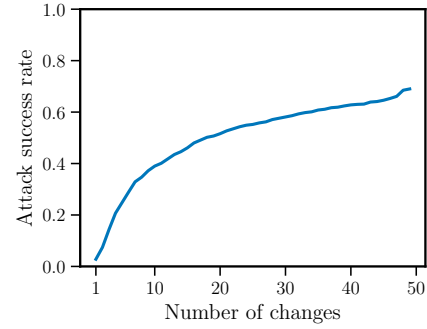


Figure 6: Even if the adversary only has access to a model trained with a different seed, adversarial examples generated on that single model still transfer to the target model. While we observe a decrease in transferability compared to Figure 3, the attack remains effective.

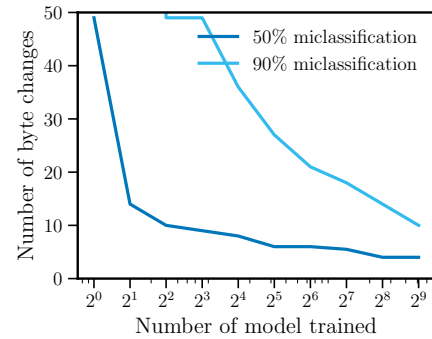


Figure 7: The adversary can improve the transferability of their attack by training additional models. Our results demonstrate that training more models allows the adversary to significantly improve the transfer rate, approaching the effectiveness of an attack directly on the target model.

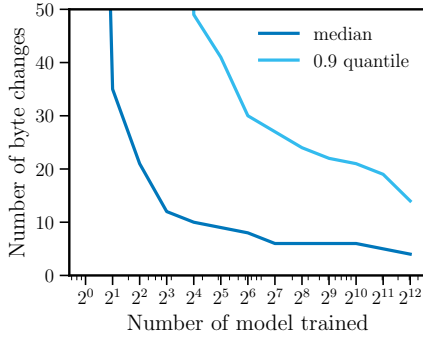


Figure 8: By training multiple models, an adversary can reduce the size of the necessary perturbation. Shown are the number of bytes an adversary must change to maintain a 50% and 90% attack success rate.

We evaluate this strategy on the dataset described in Section 4.1, using the same 1,130 samples across all experiments for consistency. An adversary who constructs evasive malware samples on one such substitute model and transfers them to another succeeds only 35% of the time with 10 bytes changed—a significant decrease from the 90% observed in the white-box setting. Figure 6 shows the complete distortion-versus-success-rate curve.

But an adversary can do better. Prior work [23, 27] has found that one of the simplest and most effective strategies to produce transferable adversarial examples is to generate an input that fools multiple white-box models simultaneously, and then attempts to transfer the attack to the (unknown) remote model. Figure 7 illustrates a scenario where the adversary trains multiple models with varying seeds on the data distribution. Adversarial examples are then generated based on a combination of these models, with a final check against the target model. While a small number of similar models requires substantial changes to evade classification, increasing the number of trained models dramatically simplifies the problem. Interestingly, even a single additional model exhibits significant transferability. Therefore if we use this approach, even adversaries without extensive compute resources can bypass detection by either making larger file modifications or attempting multiple smaller modifications.

5.2 Different Model Architecture

Although adversarial examples can be transferred between models, adversaries often have limited computational resources. We now evaluate the effectiveness of transfer attacks under a constrained-compute setting. Figure 9 shows that if the adversary has access only to a single substitute model released by the model trainer, more modifications to the file are necessary to evade the target classifier.

To strengthen the attack under the same compute budget, the adversary can train an *ensemble of substitute models* with *diverse architectures* and search for perturbations that fool all of them simultaneously. In this setting, architectural diversity is introduced by varying the convolutional depth, kernel sizes, and layer widths of the Magika architecture (details in Appendix B). We train $N = K$

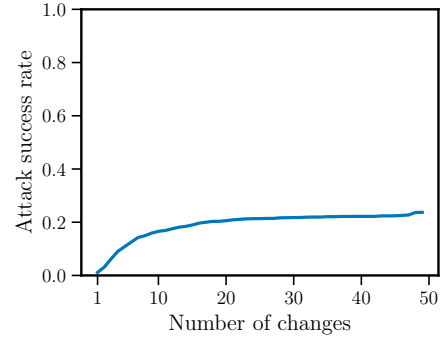


Figure 9: When the model trainer employs different model architectures for the released model compared to the target model, an adversary can still discover examples that transfer between the two. However, the success rate of such transfers is notably reduced in contrast to situations where the model architectures are the same.

such models and generate adversarial examples that optimize the combined loss across the ensemble before testing transferability to the target model. Figure 8 summarizes the results, showing that increasing the number and diversity of substitute models improves transfer success compared to the single-model baseline (Figure 7), though even this strategy cannot guarantee evasion against a defender with substantial computational resources.

5.3 Different Data Pre-processing

Deep learning models are known for their ability to learn directly from raw data, however, in many cases, such as the Magika system, model trainers make specific choices about how the data is represented or preprocessed. In Magika, for example, only certain subsets of bytes from the original file are used as input to the model. Therefore we can consider a setting where the released model and the target model use different data processing approaches. We aim to assess how effective this approach is in protecting against attacks.

We consider a scenario where the defender deploys a target model that follows the Magika architecture [15] but uses modified preprocessing offsets when selecting the input bytes. Specifically, the model extracts the first b_1 bytes starting at offset o_1 , a middle segment of b_2 bytes starting at o_2 , and the last b_3 bytes starting at o_3 , such that $b_1 + b_2 + b_3 = 1536$, the total number of bytes used by the released model. These offsets (o_1, o_2, o_3) are unknown to the attacker.

To study transferability under this defense, we trained substitute models where offsets and byte counts were *randomly sampled* from a search space that shifted each segment by up to a few hundred bytes in either direction while preserving the 1,536-byte total. Figure 10 shows that this mismatch between preprocessing pipelines significantly lowers transfer success compared to the baseline setting, when only one single model used for the transfer attacks. Then, we trained ensembles of up to 2^{12} such variants and optimized adversarial examples against their combined loss before evaluating on the defender’s model with its hidden offsets. As we can see in

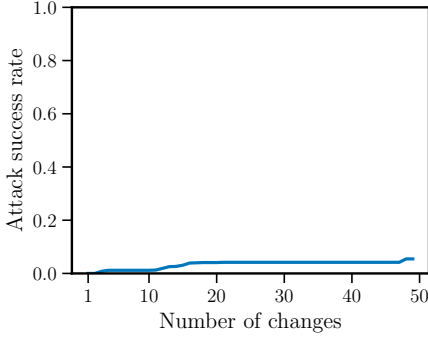


Figure 10: If the model trainer uses different data processing approaches, it significantly increases the difficulty for an adversary to directly transfer attacks between the released model and the target model.

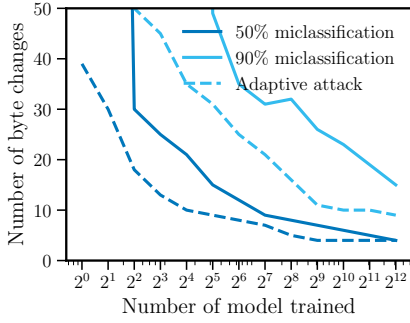


Figure 11: By training multiple transfer models, adversaries can increase the attack success rate to above 90%. In the baseline attack, the adversary trains using random hyperparameters; in the adaptive case, the adversary first identifies the best hyper-parameters used in data processing of the target model and then trains additional models with these hyperparameters.

Figure 11 as we increase the number of the models in the ensemble, the attack success rate also increases. Therefore, a well resourced adversary can defeat this defense.

We further consider an *adaptive attack* in which the adversary knows the general slicing algorithm but not the exact offsets. The attacker sends probe files with varied synthetic byte shifts and observes the target’s classification outcomes, allowing them to narrow down likely offsets. New substitute models are then trained on these candidate offsets, and adversarial perturbations are re-computed across this refined ensemble. As shown by the dashed lines in Figure 11, adaptively searching offsets reduces the compute required to find transferable examples but still incurs a higher cost than the standard transfer scenario.

A key advantage of basic defense approaches is that they generally maintain the model’s performance on clean input. This is crucial in security-sensitive applications where high false positive rates can significantly impact usability for normal users. In the next

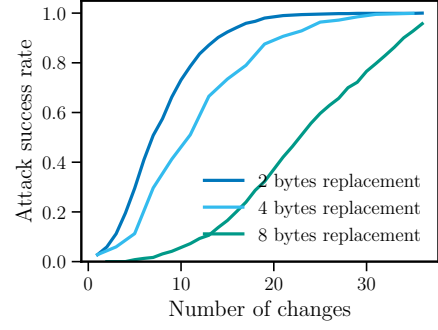


Figure 12: As we increase the perturbation bound of the attack used during adversarial training, the robustness of the resulting model increases. However, in all cases, once the attacker applies a perturbation greater than what was used during training, the attacks begin to succeed with near-100% success rate.

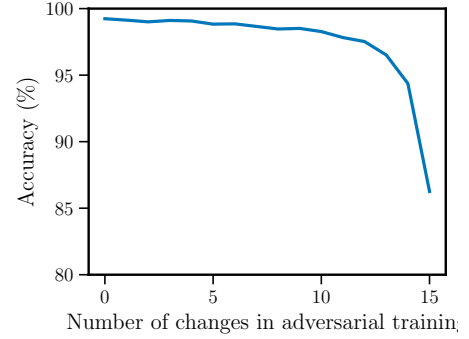


Figure 13: Increasing perturbation bound of the attack used during adversarial training reduces the accuracy of the classifier on benign data. At a perturbation bound of 0 (i.e., no attack) the accuracy is 99%; reducing this to an attack that perturbs 10 bytes marginally decreases the benign accuracy to 97%, while changing 15 bytes significantly degrades the classifier’s utility to below 90% accuracy.

section, we will explore additional defense strategies that involve modifying the training process or model architecture, which may affect the overall accuracy.

5.4 Adversarial Training

The most successful adversarial example defense is adversarial training [25], which assumes the defender knows the family of attacks that the adversary will employ and can train a model robust to these specific perturbations. For our evaluation, we reproduce this defense in the Magika setting by training a second “private” model using the same architecture as the public model, augmented with adversarially perturbed examples generated during training.

We generate adversarial examples on-the-fly in each mini-batch using our modified GCG attack (Algorithm 2) applied to the public released model. At each training step, for a clean sample x , we

compute the gradient of the loss with respect to the input bytes and greedily select coordinates to modify, one at a time, up to a perturbation budget of $k \in \{5, 10, 15\}$ bytes. For each coordinate, we evaluate candidate byte substitutions and choose the one that maximally increases the cross-entropy loss on the public model. The resulting adversarial example x' is paired with its ground-truth label and added to the batch alongside clean samples. The private model is then trained to correctly classify both x and x' by minimizing the standard cross-entropy loss on this mixed batch. We use up to 100 epoch of training and select the checkpoint with the highest average accuracy on the test set.

Figure 12 shows that this approach improves robustness for low-byte attacks (e.g., $k = 5$), but as the allowed number of modified bytes increases, adversarial examples still transfer successfully. The clean accuracy of the private model drops from 99% without adversarial examples ($k = 0$) to 97% for $k = 10$, and below 90% for $k = 15$ (Figure 13). This degradation highlights a fundamental trade-off: stronger adversarial training reduces utility in large-scale classification settings where even a small drop in accuracy impacts millions of users. Moreover, a determined adversary can replicate the same adversarial training strategy on their own substitutes, reducing the defense’s long-term effectiveness. We therefore do not further pursue this defense beyond this baseline study.

5.5 Similarity Unpairing

While adversarial training can offer robustness against specific known attacks, it often impacts benign accuracy and may not generalize well against unforeseen attack variations. Another approach for defense is aimed to reduce transfer attacks by introducing differences (randomization, architecture, preprocessing) between a potentially public model and the target deployment model as explored before. However, as shown, determined adversaries can often overcome these differences by training multiple substitute models.

Hong et al., [19] argue that the main vulnerability enabling transfer attacks is the potential *similarity* in the input-gradient landscape between the substitute model(s) available to the attacker (analogous to an open-source release or f_{θ_s} in a general scenario) and the actual deployed target model (f_{θ_o}). If gradients point in similar directions for similar inputs, perturbations crafted on one model are more likely to affect the other.

We explore **Similarity Unpairing** [19] as defense mechanism for the file classification system. The intuition is to explicitly train the public model ($f_{\theta'}$) such that while maintaining high accuracy similar to a private model (f_{θ}), its input-gradient landscape is intentionally made dissimilar to f_{θ} . This dissimilarity aims to directly disrupt the effectiveness of gradient-based transfer attacks.

Formally, starting from a trained model f_{θ} , we fine-tune it to obtain the public model $f_{\theta'}$ using a modified objective function:

$$\mathcal{L} = \mathcal{L}_{xe}(f_{\theta'}(x), y) + \lambda \cdot C_s(x, y, f_{\theta}, f_{\theta'}) \quad (2)$$

where, \mathcal{L}_{xe} represents the cross-entropy loss ensuring the model remains accurate for the primary classification task. C_s is a similarity function designed to measure the input-gradient similarity between the original model f_{θ} and the fine-tuned model $f_{\theta'}$. The hyperparameter λ balances the standard accuracy objective with the goal of minimizing gradient similarity. By minimizing C_s during

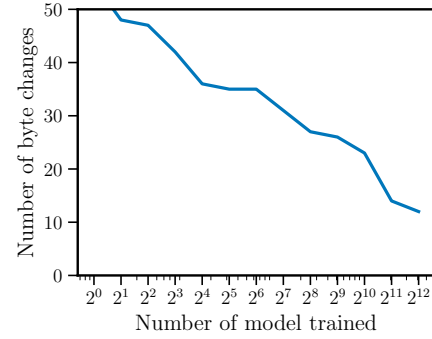


Figure 14: Increasing the number of additional model trained by unpairing approach can increase the transferability of adversarial examples.

fine-tuning, the approach "unpairs" the gradient landscapes of the two models.

For our settings adapting this approach, we would fine-tune the public model for a small number of epochs (10 epochs) using this combined loss. The specific definition of the similarity function C_s can vary; we use cosine similarity between the gradients of the two models with respect to the input x .

In the setting we consider, the adversary is aware that the defender is using the similarity unpairing approach. As an attack, the adversary trains substitute models using the same unpairing loss function to create different fine-tuned versions of the released model. First, we observed that this defense approach does not significantly affect the accuracy of the released model. Figure 14 summarizes the results for this defense against transfer attacks. As can be seen, without any additional substitute models trained by the adversary, the defense reduces the effectiveness of the transfer attack. However, as the adversary increases the number of additional models trained with unpairing, the attack effectiveness increases, similar to the previous settings; we observe a large drop in the number of bytes required to achieve a misclassification with a 90% success rate. Therefore, while this defense provides some benefit in limited settings, a resourceful adversary can likely bypass it by training additional models.

5.6 Our Proposed Defense

One challenge in defending against adversarial examples is that attackers can often estimate the preprocessing steps used and craft attacks that transfer effectively across different models. While using diverse preprocessing pipelines can help, the limited space of common preprocessing choices makes it feasible for attackers to approximate these transformations. Moreover, we saw that in the previous defense when we try to unpair the similarity of the input spaces of the model, the transfer rate is reduced. We combine the ideas from both defenses and we propose a novel defense inspired by cryptographic techniques, specifically the Advanced Encryption Standard (AES) [13].

AES is a symmetric block cipher that encrypts data using a series of substitution, permutation, and mixing operations [13]. These operations are organized into rounds, and the number of rounds

Algorithm 3 AES-Based Preprocessing Algorithm (ECB)

Require: Input data bytes D , AES encryption key K , Number of AES rounds R

Splitting:

```

1:  $n \leftarrow \text{length of } D \text{ in bytes}$ 
2:  $p \leftarrow n \bmod 16$  ▷ Calculate padding needed
3: if  $p \neq 0$  then
4:    $D \leftarrow D \parallel 0^{16-p}$  ▷ Pad  $D$  with zeros to a multiple of 16 bytes
5: end if
6:  $N \leftarrow n \div 16$  ▷ Number of 16-byte blocks

```

AES Encryption (ECB Mode):

```

7: for  $i = 1$  to  $N$  do
8:    $B_i \leftarrow D_{(i-1) \times 16 + 1 \text{ to } i \times 16}$  ▷ Extract the  $i$ -th 16-byte block
9:    $B_i \leftarrow \text{AES-Encrypt}(B_i, K, R)$  ▷ Encrypt  $B_i$  using key  $K$ 
10:   $D'_{(i-1) \times 16 + 1 \text{ to } i \times 16} \leftarrow B_i$  ▷ Store the encrypted block back into  $D'$ 
11: end for
return  $D'$ 

```

depends on the desired security level. A key feature of AES is its strong diffusion and confusion properties. Diffusion ensures that even small changes in the input propagate throughout the encryption process, resulting in significant changes in the output. Confusion obscures the relationship between the input and output, making it difficult for attackers to analyze the cipher. In our defense, we leverage these properties by incorporating a modified AES transformation into the preprocessing pipeline. However, using the full AES encryption process would make the model overly sensitive to even single-bit perturbations in the input. Therefore, we apply only a limited number of AES rounds. Our experiments demonstrate that even a single round of AES provides significant protection against transfer attacks, effectively disrupting the attacker’s ability to estimate the preprocessing pipeline and generate transferable adversarial examples. This approach introduces a dynamic and non-linear transformation that significantly expands the space of possible preprocessing functions, making it much harder for adversaries to approximate.

Algorithm 3 outlines our input preprocessing approach. Because the AES block size is 16 bytes and our model’s input size exceeds this, we split the input into multiple 16-byte blocks. AES has several modes of operation, including ECB (Electronic Codebook) and CBC (Cipher Block Chaining) [13]. We report only the results for ECB mode in the paper; we observe that CBC model has high effect on the utility of the model.

Our results demonstrate the effectiveness of incorporating AES encryption into the preprocessing pipeline as a defense against transfer attacks. First we show our approach does not heavily affect the utility of the model: as we can see in Figure 16, when we increase the number of round of AES, we see a significant drop in the accuracy. Nevertheless, we can get effective results even with one round of AES.

Figure 15 shows that our proposed defense significantly reduces the transferability of adversarial examples. In this experiment, the defender uses only one round and the adversary trains 512 models with different keys and seeds for training on the similar underlying

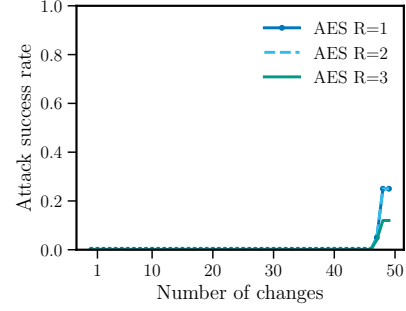


Figure 15: Our proposed defense is highly effective in reducing the transfer rate between two models trained with different keys.

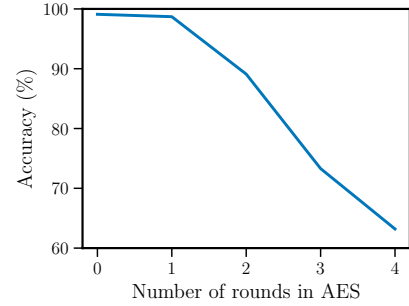


Figure 16: Increasing the rounds AES significantly reduces the utility of the final model.

dataset (different sampled from the large training data) and generate adversarial examples that fool all of them at the same time and then transfer them to the larger models. When we have a large number of modifications we can still see some misclassifications, however, this might be due to classification error and not transferability of adversarial examples. In Appendix A we discuss the adversary who can directly attack the target model and not leverage transfer attack and achieve a higher success rate than using transfer attack.

It is important to emphasize that we do not claim to defend against all transferable adversarial examples. Attacks generated from entirely unrelated models might still transfer to our target model. Our focus here is specifically on how releasing an open-source version of a model can increase the attack surface. Our findings demonstrate that if the private version of a model employs a defense similar to ours, an adversary with access to the open-source version gains no advantage in crafting successful attacks.

We finally implement this defense and, in the collaboration with Google engineers, we deploy it within the production Gmail classifier in order to improve user safety. As such, the attacks introduced in the prior section are unlikely to remain effective after the release of this paper.

Latency trade-off. In addition to classification utility, we evaluated the computational overhead introduced by the AES-based preprocessing. Our implementation processes the 1,536-byte input in 16-byte blocks using a vectorized AES routine optimized for CPU

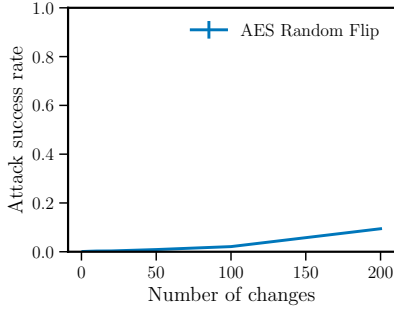


Figure 17: Our proposed approach can still fail when the adversary can change large number of bytes even at random! (averaged between 5 runs)

inference. With a single AES round (the setting we use for deployment), the additional latency per file is below a microsecond on standard production hardware, which is negligible compared to the overall inference time of the Magika pipeline. Even when increasing the number of rounds to three, the total added latency remains well under one millisecond per file, making the defense practical for large-scale, high-throughput malware scanning systems.

6 Limitations

Our evaluation focuses on transfer attacks, where the adversary does not have access to the parameters or secret key of the deployed model. In this setting, the proposed AES-based preprocessing significantly reduces transfer attacks. However, a white-box adversary with full system access could still succeed. In the Gmail proof-of-concept, a successful attack required both altering or removing the file extension and bypassing Magika as well as other rule-based classifiers. This step is specific to Gmail’s defense-in-depth pipeline and may reduce effectiveness in practice, since file handling behavior varies across operating systems and some users may be less likely to open files without familiar extensions (see Figure 17).

We again reiterate an important limitation of our work: we report attacker cost in terms of the number of bytes modified, but acknowledge this does not fully capture real-world effort—motivated adversaries could modify more bytes if needed. Also, malicious functionality was verified for the subset of files used in our proof-of-concept in isolated environments, but not exhaustively across all formats. Finally, while the defense is designed to reduce transferability from public to private models, it can still be bypassed by large-budget black-box search or direct white-box attacks.

7 Future work

This work opens up several promising directions for future research. We believe it would be interesting to refine and simplify our defense mechanisms for our task by leveraging domain specific knowledge about different file types. For example, this could involve developing specialized adversarial training approaches that exploit the unique vulnerabilities and blind spots of each file type. Alternatively, we could explore how to design adversarial training pipelines that exploit the specific characteristics of PDF files or images, and then train models to be robust against these attacks.

We also believe it would be interesting to investigate the effectiveness of our defense in other domains. Our current hypothesis is that the success of our approach in this context stems from the significant differences in patterns across various file types. By introducing diffusion and confusion through cryptographic techniques, we disrupt the attacker’s ability to transfer adversarial examples without significantly impacting the utility of the model. However, it is unclear whether this approach will generalize to harder tasks where the underlying data patterns may be more complex and subtle. Applying similar defenses in such domains could potentially lead to a significant decrease in model utility. Therefore, future work should systematically evaluate the effectiveness and trade-offs of our defense across a range of tasks and datasets.

8 Discussion and Conclusion

This paper focuses on a specific technical vulnerability – causing a file-type classifier to misclassify – but its significance lies in the system-level impact: how such a vulnerability in one component can compromise a larger security pipeline. And we show that, by leveraging recent advances in adversarial machine learning, it is possible to do this by changing just a few bytes in any given file. Fortunately, as we show, by applying relatively straightforward defense ideas, it becomes possible to increase the robustness of this classifier to attack by 75% even in scenarios where the adversary uses large amounts of compute to train additional models using the exact same pipeline.

Beyond this specific technical problem, our broader goal is to highlight the need for practical security analysis of real-world systems incorporating ML components. We aim to encourage research that moves beyond attacking isolated models to understanding how attacks manifest and can be mitigated within complex, deployed environments. This is because machine learning models are no longer isolated to single-purpose products that only classify images, or transcribe the speech in an audio file. Today’s machine learning models are sufficiently capable that they can be embedded into tools that from the outside would not give the impression there is a machine learning component in the overall system.

And because of this, for any given system, it becomes increasingly likely that there will exist some machine learning model as a component of this system. This means that instead of imagining possible applications of machine learning models and then attacking those hypothetical systems, we argue that future adversarial machine learning works should also attempt to look for real systems and then develop attacks that actually cause some specific harm. In the case of this paper, we have found that the security of the malware classifier in Gmail depends, in part, on a single neural network that is vulnerable to transferable adversarial examples. But we do not believe this will be the only system that has this property; in this way, our analysis can serve as a case study for how to evaluate the robustness of a security system that happens to contain a machine learning model and how someone might design and evaluate defenses for such systems.

Overall, we hope to encourage future work to take this same approach, and analyze end-to-end systems that contain machine learning models as subsystems which (either implicitly or explicitly) are part of the trusted computing base for the system. Furthermore,

we advocate for the development and evaluation of defenses based on their effectiveness in raising the bar for attackers in practice, contributing to incrementally more secure real-world deployments.

Ethics Considerations

We strictly followed common responsible disclosure protocols to ensure that no harm was caused in the development of our attack, and no real users were ever targeted for this work. Prior to validating our attack was effective over the production Gmail classifier we received advance permission from the Gmail team at Google, and after we completed our defense analysis we worked directly with the affected product team to develop a solution. As a result of our disclosing this attack, none of the attacks described in this work remain effective against the deployed system. We hope this research encourages more open sourcing of security research and solutions, demonstrating the positive outcomes of collaborative vulnerability discovery and mitigation. We received full approval from the affected team to publish these results.

Open Artifacts

We release the code for our defense approach alongside the main Magika GitHub repository at <https://github.com/google/magika>. However, due to the proprietary nature of the training data used in this work, we cannot share it directly. Instead, we will provide resources in collecting similar training data. All attacks used in this work are based on existing adversarial attack approaches, which are publicly available.

For ethical reasons, we opt not to publicly release our attack source code to avoid attackers leveraging it as part of real malware campaigns. Universities may reach out for access.

References

- [1] Ange Albertini. Abusing file formats; or, corkami, the novella. *The International Journal of Proof of Concept or GTF0*, (0× 07):18–41, 2015.
- [2] Ange Albertini, Thai Duong, Shay Gueron, Stefan Kölbl, Atul Luykx, and Sophie Schmieg. How to abuse and fix authenticated encryption without key commitment. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 3291–3308, Boston, MA, August 2022. USENIX Association.
- [3] Moustafa Alzantot, Bharathan Balaji, and Mani Srivastava. Did you hear that? adversarial examples against automatic speech recognition. *arXiv preprint arXiv:1801.00554*, 2018.
- [4] Giovanni Apruzzese, Hyrum S Anderson, Savino Dambra, David Freeman, Fabio Pierazzi, and Kevin Roundy. “real attackers don’t compute gradients”: bridging the gap between adversarial ml research and practice. In *2023 IEEE Conference on Secure and Trustworthy Machine Learning (SaTML)*, pages 339–364. IEEE, 2023.
- [5] Wieland Brendel, Jonas Rauber, and Matthias Bethge. Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.
- [6] Henrik Brodin, Evan Sultanik, and Marek Surovič. Blind spots: Automatically detecting ignored program inputs, 2023.
- [7] Tom B Brown. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.
- [8] Elie Bursztein and Yanick Fratantonio. Magika: Ai powered fast and efficient file type identification, February 2024. Google Open Source Blog.
- [9] Nicholas Carlini and David Wagner. Audio adversarial examples: Targeted attacks on speech-to-text. In *2018 IEEE security and privacy workshops (SPW)*, pages 1–7. IEEE, 2018.
- [10] Pin-Yu Chen, Huan Zhang, Yash Sharma, Jinfeng Yi, and Cho-Jui Hsieh. Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security (AISec)*, 2017.
- [11] Ian Darwin and contributors. *The file Command in Linux*. Linux Foundation, 2023. Available from: <https://man7.org/linux/man-pages/man1/file.1.html>.
- [12] Jacob Devlin. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.
- [13] Morris J Dworkin, Elaine Barker, James R Nechvatal, James Foti, Lawrence E Bassham, E Roback, James F Dray Jr, et al. Advanced encryption standard (aes). 2001.
- [14] Javid Ebrahimi, Anyi Rao, Daniel Lowd, and Dejing Dou. Hotflip: White-box adversarial examples for text classification. *arXiv preprint arXiv:1712.06751*, 2017.
- [15] Yanick Fratantonio, Luca Invernizzi, Loua Farah, Kurt Thomas, Marina Zhang, Ange Albertini, Francois Galilee, Giancarlo Metitieri, Julien Cretin, Alex Petit-Bianco, David Tao, and Elie Bursztein. Magika: AI-Powered Content-Type Detection. *Proceedings of the International Conference on Software Engineering (ICSE)*, 2025.
- [16] Justin Gilmer, Ryan P Adams, Ian Goodfellow, David Andersen, and George E Dahl. Motivating the rules of the game for adversarial example research. *arXiv preprint arXiv:1807.06732*, 2018.
- [17] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.
- [18] Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. *arXiv preprint arXiv:1903.12261*, 2019.
- [19] Sanghyun Hong, Nicholas Carlini, and Alexey Kurakin. Publishing efficient on-device models increases adversarial vulnerability. In *2023 IEEE Conference on Secure and Trustworthy Machine Learning (SaTML)*, pages 271–290. IEEE, 2023.
- [20] Robin Jia and Percy Liang. Adversarial examples for evaluating reading comprehension systems. *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2017.
- [21] Yulie Kwon Kim. Google workspace: A more secure alternative for today’s threat landscape. <https://workspace.google.com/blog/identity-and-security/securealternative>, 2024.
- [22] Bin Liang, Hongcheng Li, Miaoqiang Su, Pan Bian, Xirong Li, and Wenchang Shi. Deep text classification can be fooled. *arXiv preprint arXiv:1704.08006*, 2017.
- [23] Yanpei Liu, Xinyun Chen, Chang Liu, and Dawn Song. Delving into transferable adversarial examples and black-box attacks. *arXiv preprint arXiv:1611.02770*, 2016.
- [24] Sanjay Madan, Sanjeev Sofat, and Divya Bansal. Tools and techniques for collection and analysis of internet-of-things malware: A systematic state-of-art review. *Journal of King Saud University-Computer and Information Sciences*, 34(10):9867–9888, 2022.
- [25] Aleksander Madry. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.
- [26] John X. Morris, Eli Lifland, Jin Yong Yoo, Jake Grigsby, and Z. YTanjun Qig. Textfooler: A model-agnostic adversarial text generation approach for text classification. *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020.
- [27] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. In *arXiv preprint arXiv:1605.07277*, 2016.
- [28] Fabio Pierazzi, Feargus Pendlebury, Jacopo Cortellazzi, and Lorenzo 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.
- [29] Yao Qin, Nicholas Carlini, Garrison Cottrell, Ian Goodfellow, and Colin Raffel. Imperceptible, robust, and targeted adversarial examples for automatic speech recognition. In *International conference on machine learning*, pages 5231–5240. PMLR, 2019.
- [30] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.
- [31] Taylor Shin, Yasaman Razeghi, Robert L Logan IV, Eric Wallace, and Sameer Singh. Autoprompt: Eliciting knowledge from language models with automatically generated prompts. *arXiv preprint arXiv:2010.15980*, 2020.
- [32] Ilia Shumailov, Xitong Gao, Yiren Zhao, Robert Mullins, Ross Anderson, and Cheng-Zhong Xu. Sitatapatra: Blocking the transfer of adversarial samples. *arXiv preprint arXiv:1901.08121*, 2019.
- [33] C Szegedy. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.
- [34] PaX Team. Pax address space layout randomization (aslr). 2001. Available: <https://pax.grsecurity.net/docs/aslr.txt>.
- [35] F Tramèr, D Boneh, A Kurakin, I Goodfellow, N Papernot, and P McDaniel. Ensemble adversarial training: Attacks and defenses. In *6th International Conference on Learning Representations, ICLR 2018-Conference Track Proceedings*, 2018.
- [36] VirusTotal. Virustotal. <https://www.virustotal.com/>.
- [37] Huangzhao Zhang, Hao Zhou, Ning Miao, and Lei Li. Generating fluent adversarial examples for natural languages. *arXiv preprint arXiv:2007.06174*, 2020.
- [38] Andy Zou, Zifan Wang, Nicholas Carlini, Milad Nasr, J Zico Kolter, and Matt Fredrikson. Universal and transferable adversarial attacks on aligned language models. *arXiv preprint arXiv:2307.15043*, 2023.

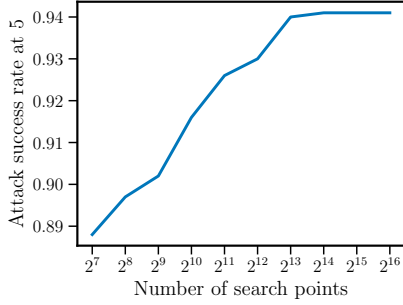


Figure 18: By increasing the number of neighbors searched in each step of the GCG optimization algorithm, we can improve the attack efficacy until we see diminishing returns at 2^{13} neighbors searched.

A Additional results

Compute vs Attack Success Rate. The GCG attack has two primary steps: first, it identifies the positions and corresponding values that change and then explore various points to find the one that maximizes the adversary objective. The number of points evaluated significantly impacts the attack’s effectiveness by increasing computational overhead. Figure 18 illustrates how varying the number of trials per step can influence attack performance. While expanding the search space can enhance attack effectiveness, this improvement is generally limited. The gradient effectively highlights crucial points, and increasing the search size beyond a certain threshold yields diminishing returns.

Real attackers do not compute gradients: Some researchers argue that in practice, adversaries do not use gradient-based approaches to circumvent machine learning defenses [4]. This could also be applicable here, where an adversary might employ fuzzing techniques to measure model accuracy and generate adversarial examples without relying on gradients. We assume such an adversary uses a similar attack strategy as GCG and the main difference is that it randomly selecting neighbors and choosing the best candidate for updates instead of computing gradients.

B Additional experimental details

To evaluate ensemble-based transfer (Figures 9–8), we trained multiple substitute models that preserve Magika’s overall design but vary in architectural details. This simulates a realistic attacker with knowledge of the general architecture but not the exact model parameters. All substitutes were trained using Magika’s preprocessing, loss function, and data methodology (Section 4.1).

We achieved architectural diversity by *randomly sampling* variations in convolutional depth (1-4), kernel sizes (8-16), and the number of fully connected layers (1-4). Each substitute architecture was required to achieve at least 95% average test accuracy.

For the attacks, we optimized adversarial examples against the combined loss of an ensemble of these substitute models, simulating a stronger attacker with more compute. We then measured the transfer success rate of these examples on the target model. The number of substitutes and compute budgets are detailed in Section 4.1.