It's Not What It Looks Like: Manipulating Perceptual Hashing based Applications

Qingying Hao, Licheng Luo, Steve T.K. Jan, Gang Wang
University of Illinois at Urbana-Champaign
Urbana, IL, USA
{qhao2,ll6}@illinois.edu,stevetkjan@gmail.com,gangw@illinois.edu

ABSTRACT

Perceptual hashing is widely used to search or match similar images for digital forensics and cybercrime study. Unfortunately, the robustness of perceptual hashing algorithms is not well understood in these contexts. In this paper, we examine the robustness of perceptual hashing and its dependent security applications both experimentally and empirically. We first develop a series of attack algorithms to subvert perceptual hashing based image search. This is done by generating attack images that effectively enlarge the hash distance to the original image while introducing minimal visual changes. To make the attack practical, we design the attack algorithms under a black-box setting, augmented with novel designs (e.g., grayscale initialization) to improve the attack efficiency and transferability. We then evaluate our attack against the standard pHash as well as its robust variant using three different datasets. After confirming the attack effectiveness experimentally, we then empirically test against real-world reverse image search engines including TinEye, Google, Microsoft Bing, and Yandex. We find that our attack is highly successful on TinEye and Bing, and is moderately successful on Google and Yandex. Based on our findings, we discuss possible countermeasures and recommendations.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning**; • **Security and privacy** → *Web application security*; *Social aspects of security and privacy*.

KEYWORDS

Perceptual Hashing; Adversarial Machine Learning; Black-box Attacks; Image Search Engine

ACM Reference Format:
Qingying Hao, Licheng Luo, Steve T.K. Jan, Gang Wang. 2021. It's Not What It Looks Like: Manipulating Perceptual Hashing based Applications. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS '21), November 15–19, 2021, Virtual Event, Republic of Korea.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3460120.3484559

CCS '21, November 15–19, 2021, Virtual Event, Republic of Korea.

ACM ISBN 978-1-4503-8454-4/21/11…$15.00
https://doi.org/10.1145/3460120.3484559

Figure 1: Example attack against perceptual hashing based image search. The results are obtained from Google's reverse image search.

1 INTRODUCTION

Perceptual hashing [29] is widely used by researchers and industry practitioners to match similar images for digital forensics and cybercrime studies [16, 26, 28, 44, 50, 55–57, 71, 72, 85, 87]. At the high level, perceptual hashing algorithms generate a fingerprint for each image so that similar-looking images will be mapped to the same or similar hash code. Unlike conventional cryptographic hashing algorithms (e.g., MD5, SHA) that generate distinct hash values for slightly changed inputs, perceptual hashing is designed to tolerate small perturbations so that slightly changed content still produces similar hash values.

Researchers have applied perceptual hashing in various applications for online abuse detection and analysis. For example, it has been used to detect malicious websites based on webpage screenshots [71], and detect counterfeit apps using app graphical UIs [55]. More recently, researchers used perceptual hashing to analyze manipulated images in misinformation campaigns [87], flag child abuse content shared online [9], and detect identity impersonation in online social networks [16]. Perceptual hashing is also heavily used in real-world reverse image search engines such as Google [18], Yandex [83], Microsoft Bing [5], and TinEye [69].

Despite the wide adoption of perceptual hashing, the robustness of the algorithms is not well understood in the context of *online abuse detection and analysis*. In practice, adversaries may exploit the weaknesses of perceptual hashing algorithms to influence dependent applications. Figure 1 illustrates an example: x_o

presents the original photo of activist Linda Sarsour (left) and Stacey Abrams (right) who was the Georgia governor candidate at that time. A few months later, an edited version of the photo $\boldsymbol{x}_m$ was posted on Facebook where the word "Communist" and the hashtag "MuslimBrotherhood" were added to the campaign sign. This manipulated image $\boldsymbol{x}_m$ went viral across the Internet shortly after posting. While the image has been cropped, resized, and altered semantically, searching $\boldsymbol{x}_m$ via Google's *reverse image search* still allows users to find the original image $\boldsymbol{x}_o$ to spot the manipulation. However, motivated adversaries can make it significantly harder for users to reverse search the edited image to find the original photo. For example, attackers can generate an attack image $\boldsymbol{x}_a$, such that $\boldsymbol{x}_a$ has a large perceptual hash distance to $\boldsymbol{x}_o$ without major visual changes. As shown in Figure 1, the original photo is no longer included in the top-ranked search results of $\boldsymbol{x}_a$.

This Paper. In this paper, we propose to examine the robustness of perceptual hashing and its dependent security applications, both *experimentally* and *empirically*. We propose a series of attack algorithms that generate attack images such that (1) the original image and the attack image are still visually similar, but (2) the attack image has a very different perceptual hash code from that of the original image. In practice, this attack can be used to target reverse image search engines and analytic tools used in cybercrime studies and online abuse analysis (like the example shown in Figure 1). To make the attack practical, we assume the attacker only has black-box access to the perceptual hashing algorithm used in the target application (i.e., without knowledge of its implementation details or parameters).

We design our attack algorithm based on a high-level observation. Fundamentally, a perceptual hashing function is a compression function to map high-dimensional data (an image) to a low-dimensional representation (hash code) to approximate human perception. However, mainstream perceptual hashing algorithms often prioritize *efficiency* and cannot perfectly mimic human perception. This is understandable considering real-world applications (e.g., a search engine) need to efficiently run image matching over billions of indexed images. To these ends, it is possible for attackers to use more sophisticated (computationally expensive) perceptual metrics to guide the optimization of adversarial noises to enlarge the hash distance without introducing significant visual changes. We design a series of attack algorithms based on this high-level idea. We also augment the attacks with additional novel designs such as grayscale initialization and image transformations to improve the attack efficiency and transferability.

Evaluation and Real-world Tests. We evaluate our attacks using both controlled experiments and empirical tests (on real-world image search engines). For controlled experiments, we focus on a standard perceptual hashing algorithm (pHash) which is widely used by security researchers. We also use a robust variant of pHash (called Blockhash) for comparison. Our experiments involve three image datasets, including ImageNet [58], a dataset of human faces [27], and a dataset of manipulated images (for misinformation dissemination) [45]. Our experiments have demonstrated the benefits of our design choices and confirmed the effectiveness of the attack against different perceptual hashing algorithms.

We then evaluate the attack against real-world reverse image search engines, including TinEye, Google, Microsoft Bing, and Yandex. We conduct a two-step experiment to (1) first compare different attack algorithms, and (2) then select the best-performing attack algorithm to test a variety of images. By manually validating the search results, we show that our attack is highly successful on TinEye and Microsoft Bing (with a 100% success rate) by completely eliminating relevant images and their websites from the first page of the search results. Yandex has a different reaction as the attack introduces significantly more false positives (to 90%). The attack is also moderately effective on Google: it succeeds on 64% of the target images while pushing the false positive rate to 66%.

Contributions. We have three key contributions:

- *First*, we formulate and design new attacks against perceptual hashing based applications in a black-box manner.
- *Second*, we evaluate our attack against the standard perceptual hashing algorithm used by security researchers as well as its robust variant on three different datasets.
- *Third*, we empirically test our attack using real-world image search engines (including TinEye, Google, Bing, and Yandex) to confirm its effectiveness.

We hope our work can lead to more research efforts to robustify perceptual hashing algorithms used by security researchers and real-world services. At the end of the paper, we have discussed possible countermeasures at both the server side and the user client side. To facilitate future works, we release our code and the annotated dataset to the research community[1].

2 BACKGROUND: PERCEPTUAL HASHING

We begin by introducing the background of perceptual hashing, followed by a literature survey on its security applications.

2.1 Perceptual Hashing and Applications

Perceptual Hashing Function. Perceptual hashing algorithms generate a fingerprint for multimedia data (e.g., images) so that similar-looking content will be mapped to the same or similar hash values. More formally, we use H to denote a perceptual hashing function. Given an input image $\boldsymbol{x}$, the function produces a binary string as the hash code: $h = H(\boldsymbol{x})$, $h \in \{0/1\}^l$. Here, $\{0/1\}^l$ represents a binary string of length l. We denote a slightly modified version of $\boldsymbol{x}$ as $\hat{\boldsymbol{x}}$ (the two images are visually similar), and denote $\boldsymbol{y}$ as a visually different image from $\boldsymbol{x}$. H is expected to meet the following requirements [88]: (1) unpredictability of hash values: $P(H(\boldsymbol{x}) = h) \approx \frac{1}{2^l}, \forall h \in \{0/1\}^l$; (2) independence of input images: $P(H(\boldsymbol{x}) = h_1 | H(\boldsymbol{y}) = h_2) \approx P(H(\boldsymbol{x}) = h_1), \forall h_1, h_2 \in \{0/1\}^l$; (3) producing the similar hash values for perceptually similar images: $P(H(\boldsymbol{x}) = H(\hat{\boldsymbol{x}})) \approx 1$; and (4) producing distinct values for different images: $P(H(\boldsymbol{x}) = H(\boldsymbol{y})) \approx 0$.

Given an input image, it typically takes three steps to compute a perceptual hash code: transformation, feature extraction, and quantization. Here, we briefly describe the high-level idea of these steps, and provide more details for popular perceptual hashing functions in Appendix A. First, the *transformation* step is to apply various

[1] https://gangw.cs.illinois.edu/hash.html

transformations (e.g., smoothing, color transformations, frequency transformations) to the image. The goal is to augment the image data for feature extraction. Second, the *feature extraction* step takes the transformed images (of size $M \times N$) to extract a feature vector of length L ($L \ll M \times N$), and/or select the most pertinent K features ($K \ll L$). Each feature in the vector is represented by p elements of type float. The feature extraction step selects global features that are resilient against normal image transformations (e.g., adding background noises). Third, the quantization step quantizes the continuous intermediate hash values to discrete values to further improve the algorithm's robustness.

Perceptual Hashing Usage in Academic Research. Perceptual hashing (e.g., pHash [29]) is frequently used by researchers to detect similar media content. We have surveyed security papers in the past 5 years to summarize key usage cases.

The most common application is to use perceptual hashing to analyze website screenshots and pair visually similar websites. This approach has been used to detect malicious (phishing) websites under domain squatting [2, 28], online survey scams [26, 71], website defacements [6], technical support scams [40], and black-hat search engine optimization (SEO) [72]. Another popular use case is to analyze displayed ads to detect fraudulent services [44, 54] or to build perceptual ad-blockers [70]. In addition, it is also used to analyze the graphical user interface (UI) of mobile apps to detect counterfeit apps [55] and detect app vulnerabilities [4, 62]. More recently, perceptual hashing is applied to analyze the images and memes distributed in misinformation campaigns [1, 38, 41, 56, 57, 64, 77, 86, 87] and detect pornography content [50, 85]. Finally, perceptual hashing has been used to match duplicated profile photos to flag identity impersonation in online social networks [16].

When using a perceptual hashing algorithm, researchers do not simply rely on hash collision to detect similar images (which can easily produce false negatives). Instead, a certain threshold is set on the normalized *Hamming Distance* [19] between two hash strings for accurate image matching. Normalized Hamming Distance measures the number of different bits between the two hash strings divided by the length of the hash string. The threshold for normalized Hamming distance varies for different applications but mostly falls between 0.1 and 0.4 [26, 54, 63, 77, 85, 86].

Perceptual Hashing based Online Services. Perceptual hashing is also widely used to build *reverse image search engines*, such as those from Google [18], Yandex [83], and Bing [5], and TinEye [69]. These reverse image search engines have been used to support applications such as catfishing detection (for online dating services) [15] and photo infringement detection [24, 52].

Relevant Existing Attacks. Researchers have explored potential attacks against perceptual hashing in the past. However, these attacks are not designed to manipulate *reverse image search*. Instead, they are mostly targeting image authentication applications. More specifically, the goal is to introduce image distortions to alter the semantic meaning of the images *without changing the image's hash value*. In this way, the perceptual hashing algorithm would not be able to detect the distortion, and falsely treat the modified image as the same as the original. For example, authors in [80] applied distortions (e.g., by changing the plate number of a car or inserting a small flag into the image) without changing the image's hash value. In [79], the authors tested both malicious modifications (e.g., adding/removing objects to/from the image) and benign modifications (e.g., rotation, cropping, JPEG compression, and additive Gaussian noises) to show that perceptual hashing algorithms cannot detect "malicious" changes effectively. Most existing attacks are manually crafted based on heuristics and cannot be applied automatically to arbitrary images.

2.2 Our Goals

In this paper, we explore attacks that aim to manipulate reverse image search based applications. Our attack goal is the *opposite* of existing attacks [79, 80] described above. More specifically, existing attacks [79, 80] aim to introduce visible distortions to that image that do not change the image hash value (to bypass image authentication systems). Our goal is the opposite: we want to introduce small/imperceptible distortions that significantly change the hash value of the image. With a highly different hash value, it would become difficult for the search engine to link the modified image to the original source. In addition, we aim to generate the attack images automatically without manual efforts.

Our goal is related to that of adversarial attacks against machine learning classifiers [11, 17, 37, 47, 48, 65] but has some differences. For example, adversarial examples aim to fool a classifier to assign a (targeted) wrong label, whereas our goal is to manipulate the similarity metric defined by a perceptual hashing function between a pair of specific images. Further discussion of our relationship with adversarial examples is in Section 9.

3 THREAT MODEL

Given an input image, the attacker's goal is to generate an attack image such that (1) the original image and the attack image are still visually similar, and (2) the attack image has a different perceptual hash code from that of the original image.

In practice, the attack targets reverse image search engines and security applications that depend on image matching. For example, for search engines, searching the attack image will no longer return the real sources (or the real sources are no longer ranked at the top of the returned results). For security applications (e.g., malicious website detection, misinformation campaign detection), the attack image can no longer be linked to (or grouped with) other relevant images, which subverts the effectiveness of these applications.

By default, we assume the attacker only has a *black-box* access to the perceptual hashing algorithm used in the target application. This means the attacker has no knowledge about the implementation details of the perceptual hashing algorithm or its configurations. We assume the attacker can query the target hashing algorithm to obtain the hash code for an image.

Later in this paper, we also explore the scenario where the attacker *does not even have the query ability* (e.g., when attacking a real-world search engine). In this case, the attacker will generate the attack image based on a local perceptual hashing algorithm to attack the target application.

4 METHODOLOGY

In this section, we describe our attack methodology. We begin with a *basic attack* against perceptual hashing algorithms. Then, with real-world applications in mind, we develop *advanced attack* methods to enhance the basic attack algorithm.

4.1 Basic Attack

Given an input image $\boldsymbol{x}_o$ and a perceptual hashing algorithm $H()$, we want to create an attack image $\boldsymbol{x}_a$ to meet two requirements: (1) *attack effect*: the hash code of the attack image $h_a = H(\boldsymbol{x}_a)$ should have a large Hamming Distance to the hash code of the original image $h_o = H(\boldsymbol{x}_o)$; and (2) *stealth and semantic consistency*: the attack image $\boldsymbol{x}_a$ should be visually similar to the original image $\boldsymbol{x}_o$.

Note that the problem formulation here is slightly different from the example described in Figure 1 as this formulation does not explicitly define $\boldsymbol{x}_m$ (the manually altered image). Among different perceptual hashing applications, some applications have an $\boldsymbol{x}_m$ (e.g., disinformation), but many do not have an $\boldsymbol{x}_m$ in their threat models such as counterfeit app detection [55] and social network impersonation detection [16]. As such, we choose this clean and yet generic formulation by only focusing on *two images* (denoted as $\boldsymbol{x}_o$ and $\boldsymbol{x}_a$). This formulation directly works for applications that do not have an $\boldsymbol{x}_m$. For applications with an $\boldsymbol{x}_m$, we can simply use $\boldsymbol{x}_m$ as the reference image (instead of $\boldsymbol{x}_o$) such that the attack image $\boldsymbol{x}_a$ will be semantically/visually similar to $\boldsymbol{x}_m$. Since $\boldsymbol{x}_o$ and $\boldsymbol{x}_m$ have a small hash distance in the first place, optimizing the hash distance to $\boldsymbol{x}_m$ is reasonable. Later in Section 5.6 and Section 6.4, we will provide concrete evaluation and examples.

Based on the above problem formulation, we have the following loss function:

$$\begin{aligned} \underset{\boldsymbol{x}_a}{\text{minimize}} \quad & \mathrm{D}(\boldsymbol{x}_a, \boldsymbol{x}_o) + c \cdot f(\boldsymbol{x}_a, \boldsymbol{x}_o) \\ \text{subject to} \quad & \boldsymbol{x}_a \in [0,1]^p. \end{aligned} \tag{1}$$

$D(\boldsymbol{x}_a, \boldsymbol{x}_o)$ denotes the function to measure the visual difference between the original image and the attack image. $f(\boldsymbol{x}_a, \boldsymbol{x}_o)$ denotes the similarity between $\boldsymbol{x}_a$ and $\boldsymbol{x}_o$ in the hash space. The key challenge is to construct a differentiable function $f()$ that allows us to run optimization. c is a scaling factor to balance the weights of the two loss terms, which is a tunable hyperparameter. p is the dimension of the input feature vectors.

Design of Hash Similarity Loss $f()$. Recall that $f()$ needs to be differentiable and reflects the similarity of the hash strings of two images. We construct a hinge-loss function that transfers the Hamming Distance of two hash strings into probabilities:

$$f(\boldsymbol{x}_a, \boldsymbol{x}_o) = \max\{\tanh(1 - |H(\boldsymbol{x}_a) - H(\boldsymbol{x}_o)|/d_t), 0\}, \tag{2}$$

where $|H(\boldsymbol{x}_a) - H(\boldsymbol{x}_o)|$ denotes the normalized Hamming Distance of the two hash strings. d_t is a parameter that denotes the target normalized Hamming distance that the attack image aims to achieve. For example, given a 64-bit hash string, if the attacker aims to cause a 20-bit difference on the hash string for the attack image, d_t should be set to $20/64 = 0.3125$. Here, we choose tanh() for f because it makes the loss function get steeper when the hash distance approaches the target d_t, which encourages the attack image to reach and even go beyond the target distance.

Choice of Visual Distance $D()$. Recall that $D()$ is used to measure the visual similarity of two images. In traditional adversarial attacks against classifiers, researchers have used L_2 distance [11]. Here, we only consider L_2 as a backup option while prioritizing more sophisticated perceptual similarity metrics. This is because L_2 distance has key limitations in describing visual differences. For example, a slightly rotated image may look the same, but the minor rotation could lead to a large L_2 change. For our attack, we primarily choose a Learned Perceptual Image Patch Similarity (LPIPS) to construct $D()$, which is shown to better mimic human perception compared to traditional metrics such as Structural Similarity Index Measure (SSIM) and L_2 [89]. Later in Section 5.2, we will use experiments to further justify our design choices.

Grayscale Initialization. Given an input image $\boldsymbol{x}_o$, a naive approach is to directly optimize the loss function on the *colored* image for all three RGB channels. However, our empirical experiments show that this naive approach is highly inefficient. To address the efficiency problem, we propose an enhancement method, called *grayscale initialization*. First, we convert the image $\boldsymbol{x}_o$ to its grayscale version $\boldsymbol{x}'_o$. Second, we run the attack on the grayscale image to generate the attack noise. The grayscale images can significantly reduce the search space, and thus improve efficiency. Third, we use the noise learned on the grayscale image $\boldsymbol{x}'_o$ as the *initialization values* for the RGB channel noises, and run the above optimization again on the three RGB channels for $\boldsymbol{x}_o$. In this way, we can significantly improve the efficiency of the attack (experimental validation is shown in Section 5.2).

Implementation Details. We implement the black-box attack described above using the Zeroth Order Optimization method described in [12]. As a derivative-free optimization method, it only needs the zeroth order oracle (i.e., the objective function in Eq. 1). By comparing the objective function values at two very close points $\mathcal{L}(x + \delta v)$ and $\mathcal{L}(x - \delta v)$ (with a small δ), we can estimate the gradient along the direction v. Under a black-box setting, back-propagation is not possible. As a result, it could take a long time to calculate the full gradient for a single image. To speed up the optimization, we apply attack-space dimension reduction [12]. The idea is to optimize the adversarial noise at a lower dimension (e.g., 32×32), and then use an interpolation method to upscale the noise to the original image size. We use ADAM's coordinate gradient descend to update a small batch of coordinates for each iteration. We set the learning rate to 0.01 to balance the convergence time and image quality.

4.2 Advanced Attack

While most researchers have been using the standard perceptual hashing algorithm (called pHash) in their work [2, 16, 26, 28, 44, 50, 54–57, 71, 72, 85, 87], online services such as TinEye may have adopted additional methods to enhance perceptual hashing to better tolerate image modifications such as cropping and resizing. For example, search engines can divide an image into smaller blocks before extracting features [59]. They can also use robust feature extraction methods such as Scale-invariant Feature Transform (SIFT) [20, 35, 81] and those based on wavelet transform (DWT) and discrete cosine transform (DCT) [14, 30, 74].

To enhance our basic attack, we introduce two additional methods. Our designs are not to target a specific perceptual hashing implementation (as we assume the target application is a black box). Instead, these designs intend to improve the transferability of the attack images across perceptual hashing algorithms.

Attack over Input Ensemble (AoE). The first method is to optimize the attack image $\boldsymbol{x}_a$ such that its hash value not only differs from that of the original image $\boldsymbol{x}_o$, but also differs from those of *slightly transformed images* from $\boldsymbol{x}_o$. By simultaneously attacking a series of similar-looking images (i.e., an ensemble), we want to improve the robustness of the attack image. The idea is inspired by a method called Expectation over Transformation (EoT) [3], which was used to improve the robustness of adversarial examples against classifiers. Here, we adapt the method to attack perceptual hashing with a new loss function. The loss function is modified from our Eq. 1 as following:

$$\begin{aligned} \underset{\boldsymbol{x}_a}{\text{minimize}} \quad & \mathrm{D}(\boldsymbol{x}_a, \boldsymbol{x}_o) + c \cdot (f(\boldsymbol{x}_a, \boldsymbol{x}_o) + \sum_{t \in T} f(\boldsymbol{x}_a, t(\boldsymbol{x}_o))) \\ \text{subject to} \quad & \boldsymbol{x}_a \in [0,1]^p. \end{aligned} \tag{3}$$

T represents a set of transformations that can be applied to $\boldsymbol{x}_o$. T can include any standard image transformations such as rotation, scaling, cropping, and brightness and contrast variations, as long as the visual changes are small. We will introduce the details for our choices of T later in Section 5. Compared with the existing EoT method [3], a key difference is that we apply the transformation t to the original image $\boldsymbol{x}_o$ instead of the attack image $\boldsymbol{x}_a$. This is for attacking search engines that have stored multiple copies (and hash values) for each of their internally indexed images.

Attack over Input Transformation (AoT). The second method also uses image transformations, but in a different way. Given an original image $\boldsymbol{x}_o$, we pick one transformation function $t \in T$ (where T is a set of transformations that introduce imperceptible changes to the image). Then we treat $t(\boldsymbol{x}_o)$ as the target to perform the *basic attack*. The attack image $\boldsymbol{x}_a$ is optimized to be further away from $t(\boldsymbol{x}_o)$ in the hash value space. The new loss function is the following:

$$\begin{aligned} \underset{\boldsymbol{x}_a}{\text{minimize}} \quad & \mathrm{D}(\boldsymbol{x}_a, t(\boldsymbol{x}_o)) + c \cdot f(\boldsymbol{x}_a, t(\boldsymbol{x}_o)) \\ \text{subject to} \quad & \boldsymbol{x}_a \in [0,1]^p. \end{aligned} \tag{4}$$

Compared with the above AoE method, a key difference is that this AoT method uses $t(\boldsymbol{x}_o)$ as the comparison target for both $D()$ and $f()$. The intuition is that the image transformation t first moves the attack image slightly away from the original image, and then optimizes the attack noise to further enlarge the hash distance. For AoT, it is possible that the choice of t will affect the attack results (which will be studied in Section 5).

5 EVALUATION

In this section, we evaluate the proposed attacks with controlled experiments. We will further evaluate the attacks against real-world image search engines later in Section 6.

5.1 Experiment Setups

We select two popular perceptual hashing algorithms that are widely used by researchers and industry practitioners.

- **Standard pHash.** pHash is a widely used perceptual hashing algorithm. We choose an open-source implementation [8] that is used by many prior works [2, 16, 26, 28, 44, 50, 54, 55, 57, 72, 85, 87].
- **Blockhash.** Blockhash (block mean value based hash) [84] is a robust perceptual hashing algorithm. It first slices an image into smaller blocks where the number of blocks is equal to the length of the hash string. Each hash bit is computed based on one block. The bit value is determined by comparing the given block with the median of this block's neighboring blocks. We choose an open-source implementation of Blockhash [36] for our experiments.

We set pHash and Blockhash with their recommended parameters. The pHash hash string has 64 bits, and the Blockhash hash string has 256 bits. To measure the differences between hash strings, we use *normalized Hamming distance*, which is the number of different bits between two hashes divided by the length of the hash. Normalized Hamming distance has a range of $[0, 1]$.

Datasets. For our evaluation, we use three different image datasets, which will be used for different purposes.

- **ImageNet.** We will use ImageNet [58] as the primary dataset for its scale and diversity. The training dataset contains 1,280,000 images from 1,000 categories. These images are mostly collected from the Internet and can be used to emulate a reverse image search engine.
- **Face Images.** We use the images of human faces from the "Real and Fake Face Detection Challenge" [27]. We only consider the real face images (1,081 in total). This dataset allows us to study related applications such as catfishing and social network impersonation detection [16].
- **Image Manipulation Dataset (IMD).** This dataset contains images collected by researchers to study image manipulations and misinformation campaigns [45]. The dataset contains "real-life" manipulated images created by real people on the Internet (also called fauxtography [77]). There are 414 original images and 2,010 manipulated images.

Experiment Methodology & Evaluation Metrics. Our experiment emulates an image searching process. Given a perceptual hashing algorithm H, we generate the hash code for each image and store them in a backend database. Then we randomly pick a set of query images $\{\boldsymbol{x}\}$ to perform a reverse image search. For a query image $\boldsymbol{x}$, we first compute its hash string $\boldsymbol{h} = H(\boldsymbol{x})$. The search function will return relevant images whose normalized Hamming distance to $\boldsymbol{h}$ is below a threshold τ.

The returned images are sorted based on their distance to $\boldsymbol{h}$. If $\boldsymbol{x}$ has a copy stored in the backend database, ideally, the search engine should return $\boldsymbol{x}$'s copy and rank it at the top. Otherwise, no result should be returned. Any irrelevant returned images (i.e., images that are not the copy of $\boldsymbol{x}$) will be treated as *false positives*.

We consider the following evaluation metrics: (1) Average false positives (FP): the average number of irrelevant images in the returned results; (2) Top-K Hit Rate (Top-K HR): the ratio of queries

Distance Function	Target hash distance d_t=0.15				Target hash distance d_t=0.31			
	Hash Dist.	L_2	Pdist	# Iterations (Gray+RGB)	Hash Dist.	L_2	Pdist	# Iterations (Gray+RGB)
L2 as $D()$	0.156	25.306	0.033	90.42+13.18	0.292	50.449	0.116	278.54+149.86
LPIPS as $D()$	0.156	6.788	0.005	22.98+1.52	0.310	20.828	0.034	87.56+1.40

Table 1: Comparison of different perceptual distance functions $D()$ in the basic attack (ImageNet).

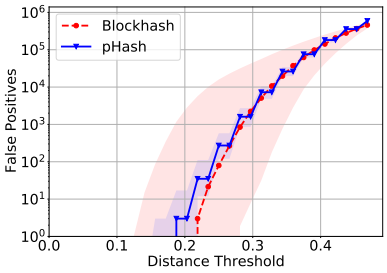


Figure 2: False positives of image queries with respect to the distance threshold. The dotted line represents the median and the colored shade represents the 5- and 95-percentiles.

where the true matching images are ranked within the top-K among the returned images; and (3) Failed Query Rate (FQR): the ratio of queries where the true matching image is not included in the returned results.

5.2 Establishing Distance Threshold

Before running the attack, we first empirically examine the proper distance threshold for image matching. First, the image searching function relies on a distance threshold τ to retrieve relevant images. Then, to launch our attack, we also need to set the desired hash distance d_t as a parameter. Intuitively, as long as $d_t > \tau$, our attack is likely to succeed. Below, we explore how to set threshold τ under normal (non-attack) cases.

Distance Threshold vs. False Positives We build a backend database using all the images from ImageNet and the Face dataset (over 1.28 million images). Then we randomly select 100 images to run search queries. By setting different distance threshold τ, we examine the number of false positives triggered by each query.

Figure 2 shows the median false positives (the dotted line) as well as the 5- and 95-percentile values (the colored shade). Because the number of false positives increases very quickly with the threshold τ, we set the y-axis in the log scale. The result shows that the robust Blockhash is more likely to trigger false positives than the standard pHash. For example, when threshold $\tau = 0.15$, most of the queries to the pHash database return no more than 1 false positive. However, certain queries to the Blockhash database already have large false positives (i.e., the 95-percentile has reached 100). To avoid excessive false positives in the search results, we want to choose the largest thresholds so that their 95-percentile number of false positives is below 100. The τ threshold for Blockhash should be lower than 0.15 (we take 0.14). Similarly, the τ threshold for pHash is set to $\tau = 0.2$. These thresholds can effectively separate different benign images apart (Appendix B shows additional analysis of the hash distances of random image pairs).

Method	Target hash distance d_t=0.15		
	Hash Dist	Pdist	# Iterations
Direct RGB (Baseline)	0.153	0.039	198.52
Grayscale+RGB (Our)	0.156	0.005	22.98+1.52

Table 2: Efficiency of attack algorithms (ImageNet): directly optimizing RGB images vs. using grayscale initialization.

5.3 Attacking Standard pHash

We next test the *basic attack* against the standard pHash and justify our design choices including the *perceptual similarity metric* and *grayscale initialization*. For this experiment, we again combine ImageNet and the Face dataset to build the backend database in order to perform image searching. Then we randomly select 50 ImageNet images and 50 face images as the target images (i.e., $\boldsymbol{x}_o$). We format ImageNet images into the size of 299×299, and format Face images into 224×224.

Impact of Perceptual Distance Function D. A key proposed design (Section 4.1) is to use Learned Perceptual Image Patch Similarity (LPIPS) for $D()$ to measure the visual difference introduced by the attack noise in the loss function. As a comparison baseline, we run the attack using L_2 distance as $D()$. We set d_t to 0.15 and 0.31 respectively where d_t represents the target hash distance we want to achieve for the attack image. For example, when d_t =0.31, it means the attack image aims to obtain a hash code that is at least 20-bit different from the original image (under a 64-bit hash).

As shown in Table 1, we report the resulting hash distance (normalized Hamming distance) for the attack images. Also, we report the resulting perceptual difference between the attack and original images, measured by L_2 distance (denoted as "L_2") and LPIPS (denoted as "Pdist") respectively. First, we observe that using L_2 and LPIPS as $D()$ can both achieve the desired attack effect. Both attacks have reached the respective target hash distance (0.15 and 0.31). Second, comparing L_2 and LPIPS, we find that using LPIPS as $D()$ creates higher-quality images with lower perceptual changes, based on both L2 and LPIPS (Pdist) measures. Finally, LPIPS achieves the desired result with fewer iterations, making the attack more efficient. Overall, the results confirm the advantage of using LPIPS as $D()$ to measure the perceptual loss.

Impact of Grayscale Initialization. Then we examine the second design choice: the grayscale initialization. In Table 2, we compare our method with a baseline where we directly optimize the attack image on three RGB channels (called "Direct RGB"). We observe that the grayscale initialization can significantly reduce the number of iterations needed to achieve the desired attack impact. For example, if we directly run the attack on the RGB channels, it takes on average 198.52 iterations. In comparison, if we perform grayscale initialization first, it takes 22.98 iterations for initialization

Dataset	Target hash distance d_t=0.15					Target hash distance d_t=0.31					Target hash distance d_t=0.47				
	Hash Dist	Pdist	# FP	Top-5 HR	FQR	Hash Dist	Pdist	# FP	Top-5 HR	FQR	Hash Dist	Pdist	# FP	Top-5 HR	FQR
ImageNet	0.156	0.005	6.72	96%	0%	0.310	0.034	3.38	0%	100%	0.441	0.120	1.68	0%	100%
Face	0.156	0.009	4.76	100%	0%	0.293	0.050	3.32	6%	94%	0.383	0.145	2.90	4%	96%

Table 3: Basic attack against pHash (the search engine distance threshold τ is set to 0.2).

and then another 1.52 iterations to generate the colored images. On average, the grayscale initialization saves 87% of the iterations.

The grayscale initialization speeds up the attack algorithm by significantly reducing the search space. In addition, perceptual hashing algorithms typically use the grayscale version of the image to compute the hashes. Noises learned from the grayscale image are likely to cause effective changes in the hash space.

Impact on the Image Searching. We next examine the attack impact on the image search results. We perform the attack with three different target hash distances: 0.15, 0.31, and 0.47 (corresponding to causing a 10-, 20-, and 30-bit difference in a 64-bit hash). According to Figure 2, we set the backend search function to use $\tau = 0.2$ to return matched images.

As shown in Table 3, when the target hash distance d_t is 0.15 and 0.31, most of the attack images can achieve the target distance. When the target distance is 0.47, the attack images on average can only get near but do not reach this target. This is not too surprising given that $d_t = 0.47$ is an extremely aggressive threshold. As shown in Figure 2, a distance of 0.47 for pHash is already far from a reasonable range (the median false positives reach 100,000). In our experiment, the search engine uses a threshold $\tau = 0.2$ to limit the number of false positives. Overall, we confirm that the basic attack can successfully generate effective attack images (when d_t =0.3 and 0.47). In these cases, the original copy of the query image is mostly hidden from the returned results, leading to a near 100% failed query rate (FQR). The top-5 hit rate (HR) is no more than 6% for all the query images.

Table 3 shows that the perceptual distance (denoted as "Pdist", measured by LPIPS) is increasing as we aim for a higher d_t, as we need to introduce more perturbations. Note that LPIPS (Pdist) has the value range of [0,1]. A pair of images are still visually similar even when LPIPS is around 0.1. Figure 3 shows example images under different target hash distances. We intentionally pick two images from two extremes in terms of the visual changes. For example, the "mushroom" image (*Agaric*) has almost no visible changes even when getting to hash distance of 0.47 (LPIPS = 0.030). The "bird" image (*Robin*) has more visible changes but the changes are acceptable even at 0.31 (LPIPS = 0.097). Overall, our results confirm that the basic attack can successfully generate attack images to manipulate the reverse image search.

5.4 Attacking Blockhash

After experimenting with the standard pHash, we next attack the *Blockhash* algorithm which is a more robust variant of pHash. In this experiment, we start with the *basic attack*. Considering that Blockhash is more likely to produce false positives (see Figure 2), we set slightly smaller target thresholds for d_t (0.15 and 0.31).

As shown in Table 4, the basic attack can successfully generate attack images that reach the target hash distance and effectively avoid the original copies being returned. When the target hash

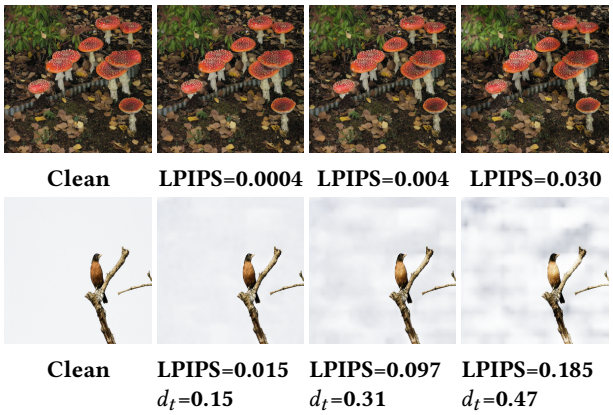


Figure 3: Example images for the basic attack against pHash.

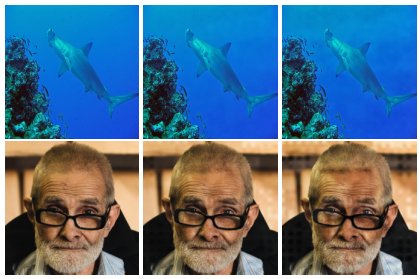


Figure 4: Example images for the basic attack against Blockhash with different target distance d_t to the original image.

distance d_t is set to 0.31, ImageNet and Face images can reach an average hash distance of 0.314 and 0.313 respectively. Across all the attack settings, we can completely hide the original copies from returned results, with a 100% FQR and a 0% top-5 hit rate ($\tau = 0.14$, as configured in Section 5.2). Compared with attacking the standard pHash (see Table 3), attacking Blockhash indeed requires larger visual changes, i.e., leading to higher perceptual changes in the attack images (a higher Pdist). We show the example attack images for the Blockhash attack in Figure 4. Overall, the attack images are still visually similar to the original images (the Pdist is still at the 0.1 level). These results confirm that our attack is effective on the more robust Blockhash algorithm too.

5.5 Transferability Evaluation

So far, we have experimented with the attack setting where we query the target hash function to generate the attack images. In practice, this query ability is often unavailable, and we will need to rely on *transferability* to generate attack images based on a local hash function. Next, we run experiments to compare the *basic attack* and the *advanced attacks* in terms of attack transferability.

Dataset	Target hash dist d_t=0.15					Target hash dist d_t=0.31				
	Hash Dist	Pdist	# FP	Top-5 HR	FQR	Hash Dist	Pdist	# FP	Top-5 HR	FQR
ImageNet	0.149	0.016	0.02	0%	100%	0.314	0.049	0	0%	100%
Face	0.149	0.038	4.08	0%	100%	0.313	0.119	0	0%	100%

Table 4: Basic attack against Blockhash (the search engine distance threshold τ is set to 0.14).

Attack Method	pHash, Target d_t=0.31		Blockhash, Target d_t=0.15	
	Hash Dist	Pdist	Hash Dist	Pdist
AoE	0.283	0.082	0.150	0.049
AoT1	0.239	0.123	0.150	0.122
AoT2	0.239	0.157	0.160	0.164
AoT3	0.244	0.131	0.155	0.122

Table 5: Advanced attacks against pHash and BlockHash using Face dataset (AoT1: cropping, AoT2: rotation, AoT3: disproportionate scaling).

Transformation Functions used in Advanced Attacks. We first introduce the transformation functions $T = \{t\}$ to be used in advanced methods. At the high level, any standard image transformation, as long as it does not significantly temper with the semantics of the image, can be used. For this experiment, we design three transformations that can largely preserve image semantics:

- **Cropping (t1)**: remove 2.5% at the edge of an image.
- **Rotation (t2)**: rotate an image by a small angle ($0.02 \times 360 = 7.2$ degree), and crop 5% of the image to remove the black edges introduced by the rotation.
- **Disproportionate Scaling (t3)**: slightly stretch an image by increasing the image width to 1.1 times of the original width. Then we crop the image back to its original size by removing excessive width at the edge.

For advanced method AoE, all three transformations are used. For AoT, it can choose any of the three transformations to perform the attack. We call the three AoT attacks as AoT1, AoT2, and AoT3, respectively. We have confirmed that these transformations, when applied alone (without running our attack), cannot effectively create meaningful hash distance for the images. For example, under pHash, the average pHash distance caused by the above three transformations is 0.095 with a standard deviation of 0.039 on the ImageNet testing set. These transformations alone are far from achieving the desired attack impact (e.g., d_t=0.31).

Running Advanced Attacks. We first run the advanced attacks against pHash and Blockhash to establish a baseline. The results are shown in Table 5. In general, the advanced attacks can successfully reach the target hash distances. The result is slightly worse for pHash when $d_t = 0.31$ (the resulting hash distance is within 0.239–0.283). In the meantime, the produced attack images also have larger visual changes (Pdist is at the 0.1 level). The example images of the advanced attacks are shown in Figure 5.

Transferability Experiment. To explore transferability, we run the basic and the advanced attacks against pHash, and then examine the resulting hash distance of the attack images *under different hashing functions*, including pHash, Blockhash as well as other generic hashing algorithms: aHash (average hash), wHash (wavelet hash), and dHash (difference hash). At the high-level,

Clean AoE AoT1 AoT2 AoT3

Figure 5: Example images for the advanced attacks against Blockhash with target hash distance d_t=0.15.

Attack Method	Transferred Distance (target hash dist $d_t = 0.31$)				
	pHash	Blockhash	aHash	dHash	wHash
pHash Basic	**0.313**	0.122	0.073	0.164	0.081
pHash AoE	**0.324**	0.158	0.105	0.215	0.109
pHash AoT3	**0.246**	0.163	0.100	0.200	0.103
Attack Method	**Transferred # FP (target hash dist $d_t = 0.31$)**				
	pHash	Blockhash	aHash	dHash	wHash
pHash Basic	**3.38**	2.6	74.12	15.34	22.66
pHash AoE	**2.64**	1.82	80.74	9.28	12.44
pHash AoT3	**3.2**	1.48	130.54	12.4	14.08
Attack Method	**Transferred Top-5 HR (target hash dist $d_t = 0.31$)**				
	pHash	Blockhash	aHash	dHash	wHash
pHash Basic	**0%**	68%	36%	46%	30%
pHash AoE	**0%**	36%	20%	24%	8%
pHash AoT3	**10%**	28%	16%	24%	12%
Attack Method	**Transferred FQR (target hash dist $d_t = 0.31$)**				
	pHash	Blockhash	aHash	dHash	wHash
pHash Basic	**100%**	32%	58%	54%	66%
pHash AoE	**100%**	64%	78%	76%	90%
pHash AoT3	**88%**	72%	78%	74%	84%

Table 6: Transferring the attack image optimized for pHash to other hash functions (ImageNet). We show average hash distance, average number of false positives (FP), top-5 hit rate and failed query rate (FQR) under each hash function.

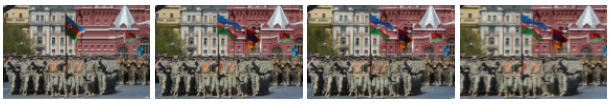
aHash computes the average of image pixel values; wHash uses Discrete Wavelet Transformation (DWT) to extract features to compute the hash code; dHash computes hashes by comparing adjacent pixels. We include more detailed descriptions of these hashing algorithms in Appendix A.

To fairly compare the searching results for different hashing algorithms, we need to determine their own distance threshold τ for the backend database. Like before, we control the false positives, and choose the thresholds for different hash algorithms so that their 95-percentile false positives are all under 100. Following the same methodology of Section 5.2, we set τ for aHash, dHash and wHash as 0.05, 0.16 and 0.05, respectively. We notice that these thresholds are lower than that of pHash (τ=0.2) because aHash, dHash and wHash are more likely to produce false positives.

The results are presented in Table 6 for ImageNet. For AoT, since we did not observe major differences between the three variants, we only show the results for AoT3 (disproportionate scaling) for brevity.

Attack Method	Target Hash Dist d_t=0.31	
	Hash Dist	Pdist
pHash Basic	0.30	0.027
pHash AoT3	0.244	0.146

Table 7: Basic attack and AoT3 attack on IMD dataset.



Clean Manipulated Basic AoT3

Figure 6: IMD pHash attack example. The manipulated image is created by vandalizing a flag in the original image and adding a new flag (it differs from the clean image by 0.125). After our attack, the resulting hash distance reached 0.31 for the basic attack and 0.25 for the AoT3 attack.

As shown in Table 6, when the attacker uses a local *pHash* to run the attack, the resulting attack images perform well with respect to pHash distance (getting close to 0.31) and can effectively hide the original copies from searching results (0%–10% Top-5 hit rate and 88%–100% FQR). When using the image to attack the searching function built on other hashing algorithms (namely, Blockhash, aHash, dHash, and wHash), the "transferred" attack impact is reduced.

Comparing different attack algorithms, Table 6 confirms that advanced attacks (AoE and AoT3) are indeed more transferable than the basic attack. As shown in Table 6, AoT3 obtains a 88% FQR on pHash (indicating the attack successfully hides the true matching images for 88% of the query images). Under a transferring attack against other hashes, the FQR is still maintained at a high level (74%–84%). In comparison, the transferability of the basic attack is weaker with a lower FQR (30%–68%). A similar trend can be observed for the top-5 HR metrics. We also find the performance of AoT3 and AoE are comparable across different hashing algorithms. Note that AoE is three times slower than AoT3 (with $|T| = 3$). As such, AoT3 is a better choice.

Overall, these results suggest that the transferred attack is still effective using advanced attack methods.

In practice, attackers may jointly optimize the attack noise against multiple hashing algorithms simultaneously. This idea has been explored in ensemble-based adversarial examples in the classifier setting [34]. Further discussions are presented in Section 7.1.

5.6 Evaluation with IMD Dataset

Finally, we perform a quick evaluation with the Image Manipulation Dataset (IMD), as a case study. Recall that IMD contains manually manipulated images collected from the Internet. In total, the dataset contains 2,010 image pairs where each pair contains one original image and one manipulated image. By comparing the pHash and Blockhash distances of such image pairs, we find that over 50% image pairs have a distance below 0.2. For these image pairs, a reverse image search engine can easily find the original image by searching the manipulated one. Image pairs that already have a large hash distance (e.g., > 0.2) typically involve drastic changes (e.g., cropping 90% of the original images). An example image is shown in Appendix C.

For the rest of the image pairs with a pHash distance below 0.2, we run our attack to further enlarge their hash distance. This will help to eliminate the original images from the search results. As a quick experiment, we run the basic attack and the AoT3 attack against pHash. Given a pair of original and manipulated images $(\boldsymbol{x}_o, \boldsymbol{x}_m)$, we aim to generate an attack image $\boldsymbol{x}_a$ such that it looks similar to the manipulated image $\boldsymbol{x}_m$ while enlarging the pHash distance (from $\boldsymbol{x}_m$) to reach 0.31. Table 7 shows that the attack is successful. The attack images (especially the basic attack) can successfully reach the target hash distance with imperceptible visual changes. An example image is shown in Figure 6.

6 REAL-WORLD EXPERIMENTS

So far, we have demonstrated the attack effectiveness using controlled experiments. In this section, we further run empirical tests with real-world reverse image search engines.

6.1 Methodology Overview

Challenges and Approaches. The key challenge of the real-world experiment is that it cannot be fully automated. First, unlike the controlled experiment where there is only one copy of the query image in the database, real-world search engines often return *multiple* websites that have indexed the query image or its legitimate variants (e.g., resized or cropped versions). To determine true positives and false positives (i.e., irrelevant images), we need to manually validate the search results. Here, we cannot use any perceptual metrics to determine true/false positives automatically, precisely because perceptual hashing is our attack target.

The needs for manual inspection limit the scale of our experiment. To cover a wide range of attack algorithms and target images (with reasonable manual efforts), we divide the experiment into two stages. *First*, we use a small set of target images to compare a large number of attack algorithms. *Second*, we select the most effective algorithm from stage-1 and apply it to a larger set of target images. We use the Face dataset [27] for this experiment. This dataset contains face images of models (e.g., used for online advertising and promotions), and thus the images are already indexed by different websites in various forms. For example, we searched the 50 clean face images used in Section 5 with Google search, and the average number of returned entries is 108.5 (the first quartile is 26 and the third quartile is 178). The results confirm that the Face images are already heavily indexed.

Search Engines. We select four popular reverse image search engines, including three free services (Google, Microsoft Bing, and Yandex) and one commercial service (TinEye). For TinEye, we purchased its search API access for our experiment. Given a query image, the search engines would return a list of entries where each entry contains a matched image and the website (URL) that hosts the image. For example, Google has a section called "*Pages that include matching images*" to display the search results (as shown in Figure 1). Other search engines have similar sections. Note that social medial platforms such as Flicker and Pinterest also have image search functions, but their search engines are more limited. For example, their search functions are focused on "relevant images" (images that contain similar types of objects) instead of finding the

	TinEye							Google						
Evaluation Metric	**Original**	**Blockhash**			**pHash**			**Original**	**Blockhash**			**pHash**		
		AoT3	AoE	Basic	AoT3	AoE	Basic		AoT3	AoE	Basic	AoT3	AoE	Basic
Avg. Reduction Rate	N/A	60%	20%	24%	65%	30%	5%	N/A	62%	45%	40%	70%	64%	50%
Top1 Hit Rate	100%	60%	80%	80%	60%	80%	100%	100%	60%	100%	100%	80%	80%	80%
Top10 Hit Rate	100%	60%	80%	80%	60%	80%	100%	100%	80%	100%	100%	100%	100%	100%
False Positive Rate	0%	40%	20%	20%	40%	20%	0%	2%	33%	7%	11%	13%	13%	12%
	Bing							**Yandex**						
Evaluation Metric	**Original**	**Blockhash**			**pHash**			**Original**	**Blockhash**			**pHash**		
		AoT3	AoE	Basic	AoT3	AoE	Basic		AoT3	AoE	Basic	AoT3	AoE	Basic
Avg. Reduction Rate	N/A	100%	100%	100%	100%	100%	100%	N/A	-611%	-871%	-756%	-813%	-1086%	-1014%
Top1 Hit Rate	80%	0%	0%	0%	0%	0%	0%	100%	80%	80%	80%	40%	80%	100%
Top10 Hit Rate	80%	0%	0%	0%	0%	0%	0%	100%	80%	80%	80%	40%	80%	100%
False Positive Rate	20%	100%	100%	100%	100%	100%	100%	0%	39%	56%	58%	76%	92%	72%

Table 8: Real-world experiments with attack images optimized for pHash (d_t = 0.31) and Blockhash (d_t = 0.15) using different attack algorithms. "Avg. Reduction Rate" shows the average rate of search result reduction caused by the attack image (in comparison with the original image).

modified copies of the query image. Also, they only index images of their users. For these reasons, we did not consider Flicker or Pinterest in this experiment.

Ethical Considerations. Testing cloud APIs with adversarial images has been a common approach adopted by researchers [7, 13, 22, 47, 48, 60, 82, 85] and our experiments follow a similar setup. We have carefully controlled the number of queries and the query rate to avoid overwhelming the search engines. In addition, after the experiments, we manually checked the searching results again to make sure that none of the attack images were indexed by the search engines. More specifically, after the experiment at a given search engine, we searched each $\boldsymbol{x}_o$ again and made sure the corresponding attack images x_a were not in the returned results. Then, we directly searched x_a, and also did not find any archived copy of x_a. If x_a were indexed by the search engine, directly searching x_a should find it out. We performed another sanity check (by repeating the above searches) three months after the experiment and confirmed the attack images were not indexed.

6.2 Small Experiments on Search Engines

As the *first* step, we use a small set of images to test a large number of attack algorithms. We select 5 random images from the Face set. For each image, we run six attack algorithms including the Basic, AoE, and AoT3 attacks optimized against pHash, and the Basic, AoE, and AoT3 attacks optimized against Blockhash. Although the number of the target images is small, the experiment still involves significant manual efforts to validate all the searching results. For each query, we manually annotated the top-50 returned entries from four search engines (1,236 returned entries in total).

Comparing Attack Algorithms. As shown in Table 8, we report the top-1 and top-10 hit rate, and the average false positive rate. Since we only manually validate the top-50 returned entries, Failed Query Rate (FQR) is not applicable here. We introduce a new metric called *average reduction rate*, which measures the rate of search result reduction caused by the attack image. Suppose a search engine returns n_o images for the "original image" and returns n_a images for the "attack image", the reduction rate is $(n_o - n_a)/n_o$. We then average the reduction rate across query images.

Overall, Table 8 shows our attack is still effective on most search engines, but the attack impact is reduced compared to that of the controlled experiments (Section 5). To better explain the results, we use *TinEye* as a walk-through example. First, the "original" column shows that TinEye's search results are highly accurate for non-attack images. Both top-1 and top-10 hit rates are 100% with a 0% false positive rate. This indicates that all the returned results from TinEye are correctly matched with the query images.

Second, under the attack images, TinEye's top-1 and top-10 hit rates are all dropped. For example, the top-10 hit rate for AoT3 is 60%, meaning that for 40% of the query images, the true matching images are pushed out of the top-10 entries (i.e., first page). Given users usually do not click on the search results beyond the first page [25], we consider the attack is effective. Third, TinEye's false positive rates go up to 20%–40% (from 0%). This means the attack not only decreases the rankings of the true matching images, but also introduces more irrelevant images to distract users. Fourth, the average reduction rates for attack images are within the range of 5%–65%. This confirms our attack also reduces the overall number of returned results in TinEye.

Comparing the six attack algorithms, *AoT3* achieves better results (AoT3-pHash and AoT3-Blockhash are comparable). This is consistent with Section 5.5 (where we show AoT3 has a better transferability). This conclusion is true for all four search engines.

Comparing Search Engines. The attack works differently on the four search engines. For Google and TinEye, our attack works on certain images but not all of them. We have manually examined the failed cases but did not find common patterns among those images (see Appendix E). Later in Section 6.3, we will slightly tune up the attack magnitude to run the experiments again with a larger set of images. On Bing, our attack is very successful. Bing could not find any true matching images for the attack images. Both the top-1 and top-10 hit rates are 0%, and the false positive rates are 100%. Yandex has a unique reaction to our attack where the average reduction rate is negative (ranging from -600% to -1000%). It turns out our attack has increased the number of returned entries by 4–10 times, by introducing major false positives to distract users. For example, for AoE-pHash, the false positive rate gets to 92%.

Evaluation Metrics	TinEye	Google	Bing	Yandex
Avg. Reduction Rate	100%	88%	100%	-1364%
Top1 Hit Rate	100%→0%	88%→36%	34%→0%	82%→58%
Top10 Hit Rate	100%→0%	96%→36%	34%→0%	86%→58%
False Positive Rate	0%→100%	11%→66%	66%→100%	21%→90%

Table 9: Real-world experiments with attack images optimized for pHash (d_t = 0.34). We show results in the format of "original → attack".

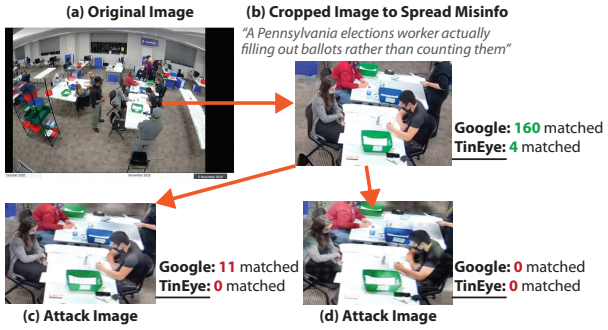


Figure 7: Example of a voting fraud image.

6.3 Increasing the Experiment Scale

As the *second* step of the experiment, we now focus on the best attack algorithm (AoT3-pHash) and test more images (50 random images from the Face set). Recall that Table 8 shows that directly testing the attack images from the controlled experiments is not as strong as before (possibly due to the search engines' countermeasures). Realizing this, in this experiment, we slightly tune up the attack thresholds for AoT3-pHash. We increase the target hash distance d_t from 0.31 to 0.34 (the quality of images is still acceptable) and increase the disproportionate scaling factor from 1.10 to 1.24. We again manually validate the top-50 returned entries for each query from four search engines (2,578 returned entries in total).

Table 9 shows that after slightly tuning up the attack magnitude, the attack becomes highly effective. More specifically, on TinEye and Bing, the attack has successfully eliminated *all* of the true positives (true matching images) from the returned results. Their top-1/top-10 hit rates are reduced to 0% and the false positives reach 100%. This means all the returned results are irrelevant. On Yandex, the attack introduces 1364% more returned images and 90% of them are false positives (i.e., distracting users from spotting the true matching images).

The attack is also more successful on Google than before. The top-10 hit rate is 36%, which means the attack is successful on 64% of the query images, hiding their true matching images away from the first page (top-10). In the meantime, the false positive rates are increased from 11% to 66% by the attack images. Finally, the average reduction rate is 88%, suggesting that 88% of the previously returned results are now eliminated.

6.4 Case Study

Finally, we demonstrate a case study to show the real-world implication of the attack. Due to space limit, we focus on one recent event as an example. On November 6th, 2020 (two days after the 2020 U.S. presidential election), a tweet shared on social media claimed that the ballot counters in Pennsylvania were seen in the act of committing voter fraud. Figure 7-(b) (which is a screen capture of a video) went viral on social media with the tweet. The caption of the image suggested that "*a Pennsylvania elections worker actually filling out ballots rather than counting them*".

This turned out to be a false claim. According to Inquirer's report [73], the ballot counter in the image was transcribing ballots that had been damaged and were unable to be fed into the counting machines. The ballot counter was following the required procedure to perform this task. More importantly, in the original video footage, the ballot counter was being observed by poll watchers from both political parties (Figure 7-(a)). To push the false narrative, the image/video was manipulated by cropping out the poll observers nearby (i.e., creating a "zoomed-in" version).

Through reverse image search, an Internet user can easily find the sources of the modified image and the related "fact-checking" information. For example, searching the top-right image on Google and TinEye returned 160 and 4 entries respectively (both results include the Inquirer report [73]). To emulate a potential adversary who wanted to make it harder for users to find the original source of this image, we apply the attack algorithms to generate attack images (which could have been used to spread misinformation). Attack image Figure 7-(c) is generated by AoT3-pHash (d_t=0.28, scaling factor=1.2). We find that it is already effective on TinEye (by hiding all the matched images). In the meantime, this image reduced Google's returned entries from 160 to 11. For attack image Figure 7-(d), we tune up the attack using AoT3-Blockhash (d_t=0.22, scaling factor=1.3), and the attack image successfully reduced the number of returned entries to 0 for Google too. In this way, it becomes harder for users to find relevant "fact-checking" information by reverse image searching.

7 DISCUSSION OF POTENTIAL DEFENSES

So far, we have demonstrated the effectiveness of our attack in both controlled experiments and empirical tests. Before discussing countermeasures, we want to first dive deeper to reason why the attack works. If we revisit the primary loss function in Eq. 1, we can see the attack is essentially exploiting the misalignment between $D()$ (human perceptual difference) and the $f()$ (an approximated hash similarity). In other words, the hash value similarity is not perfectly aligned with human perceptual similarity for images. In our attack, we use the LPIPS for $D()$ to approximate human perception, which is a recently proposed perceptual metric that outperforms existing metrics [89]. Unlike computing a perceptual hash (which can be done directly), LPIPS incurs significantly higher costs (in terms of time and image datasets) because LPIPS requires a training stage. Even if we exclude the training cost, LPIPS is still slower to compute in runtime. We argue that the attacker has the luxury to use an expensive perceptual metric to optimize the attack noise because they only focus on certain images of interest. As a comparison, image search engines are mostly dealing with *normal search queries* against billions of indexed images [69]. Search engines need to balance efficiency (query latency) with searching robustness. From this perspective, there is a power imbalance between attackers and defenders.

Dataset	Hash Distance					Pdist	Searching Results		
	pHash	Blockhash	aHash	dHash	wHash		#FP	Top-5 HR	FQR
ImageNet	0.368	0.327	0.366	0.438	0.386	0.109	2.88	0%	100%
Face	0.371	0.317	0.354	0.421	0.375	0.146	3.38	0%	100%

Table 10: Adaptive attacks against an ensemble of hashes (d_t=0.31) on ImageNet and Face datasets.

Experiment Setup and Results. We again use the ImageNet and Face datasets for this experiment. We set the target hash distance d_t=0.31 for the adaptive attacker.

7.1 Server-side Robust pHash

Regarding countermeasures, one direction is to develop more robust perceptual hashing algorithms for the server-side image searching. For example, one possible direction is to improve robustness by *adding redundancy*. The server side may consider an ensemble of a diverse set of hashing algorithms to robustify the reverse image search. However, attackers can also jointly optimize their attacks against an ensemble of hashing algorithms (which turns this into a cat-mouse game).

Another challenge for the server-side solution is to maintain the high efficiency of searching and the high quality returned results for *normal* (non-attack) queries. Considering normal queries are likely the majority, one idea to make the trade-off is to allow users to enable/disable a robust search feature on demand. For example, if a user is trying to investigate a suspicious image in the news, they can turn on the robust feature, allowing the search engine to be more permissive on the returned results. In this case, users can tolerate more false positives in order to locate the original images. During normal searches, users can turn the robust feature off to receive highly relevant returned results. This could enable the server-side to adopt more aggressive/robust countermeasures. We leave further exploration of this idea to further works.

As a preliminary exploration, we perform a quick experiment to examine the server-side defense idea. More specifically, we first construct a searching database using an ensemble of hashing algorithms at the server side. Then we explore the capability of an *adaptive attack*.

Server-side Hash Ensemble. Suppose the server-side use multiple hashing algorithms to robustify the search. For each image, the server computes multiple hash codes for this image in the backend database. For this experiment, we assume the server uses pHash, Blockhash, aHash, dHash, and wHash.

When receiving an incoming search query, the server first independently uses each hashing function to produce a list of matched entries. The distance threshold τ for each hashing algorithm is set the same way as in Section 5. Once the server gets the five lists of returned entries, we perform a simple round-robin arbitration to merge the five lists into a single ranked list. The idea is to iteratively take the top-ranked item from each list and add it to the merged list (as long as the item is not already on the merged list).

Co-optimizing Attack Noises. The adaptive attacker co-optimizes the attack noises against pHash, Blockhash, aHash, dHash, and wHash simultaneously. Given an input image $\boldsymbol{x}_o$, the attacker aim to produce an attack image $\boldsymbol{x}_a$ such that $\boldsymbol{x}_a$ differs from $\boldsymbol{x}_o$ under each of the 5 hashing functions for at least d_t. The new loss function for the adaptive attacker is the following:

$$\begin{aligned} \underset{\boldsymbol{x}_a}{\text{minimize}} \quad & \mathrm{D}(\boldsymbol{x}_a, \boldsymbol{x}_o) + c \cdot \sum_{f \in F} f(\boldsymbol{x}_a, \boldsymbol{x}_o)) \\ \text{subject to} \quad & \boldsymbol{x}_a \in [0,1]^p. \end{aligned} \tag{5}$$

where F denotes the set of hashing algorithms considered by the attacker.

Table 10 shows the experiment results. We find that after co-optimizing the noises, the produced attack images achieve the desired hashing distances under all five hash functions. Interestingly, even though d_t is set to 0.31, the resulting hash distances under dHash, wHash, aHash and pHash are significantly higher than 0.31. This is likely due to the need for achieving the desired distance for Blockhash simultaneously (which is a more robust hashing algorithm). The visual differences introduced by noises are slightly higher compared with attacking a single hash (see Section 5). Even so, the Pdist is still at the 0.1 level, and the attack images are of good quality. Finally, we search the attack images at the server-end with multiple hashing algorithms. We observe the co-optimized attack images are highly successful, producing 0% top-5 hit rates and 100% FQR. The result indicates that a simple ensemble of multiple hashing algorithms at the server side is insufficient against adaptive attacks. Further research is needed to robustify the searching at the server end.

7.2 User-side Robust Search

Another direction is to develop user-end solutions, considering that not all the search engines are willing to (or able to) implement robust server-side solutions in the short term. In comparison, a client-side solution can be immediately implemented as a browser extension to benefit users. Our idea is help users to pre-process images before searching them. Given an image $\boldsymbol{x}$, we can automatically generate a small set of transformed images $\{\boldsymbol{x}'\}$ that are visually similar to the original image. The set of images $\{\boldsymbol{x}'\}$ are generated with randomized parameters for each given image (and for each user client) such that adversaries cannot pre-optimize an attack image that is *universally effective against all the users on the Internet*. For each image $\boldsymbol{x}$ that a user wants to search, the browser extension issues multiple search queries for $\{\boldsymbol{x}'\}$ to obtain the parallel (or fused) search results.

Due to space limit, we put our preliminary experiment with this idea in Appendix D. As a short summary, we implemented a simple version of this idea and tested it on the attack images generated from Section 6.3 (i.e., the strongest attack images). We perform the searches on Google and show that the defense method can improve the top-1 and top-10 hit rates while reducing false positives. Our experiments are still preliminary — future works will be focusing on designing more effective methods to generate the search image

set $\{x'\}$, and better fusion methods to consolidate multiple lists of returned results.

8 LIMITATIONS AND FUTURE DIRECTIONS

Our paper has a few limitations. First, our attack method can be potentially improved further by using more sophisticated optimization functions for gradient estimation. We leave such exploration to future works given the current method is sufficiently effective to achieve the attack goals. Second, our evaluation scale, especially for the real-world experiments, is not very high. The scale of the real-world experiments is limited by the needs for *manual inspection* of all the search results. Given the consistency of results across experimental settings, we believe the evaluation is sufficient to support our conclusions. Third, we have experimented with three datasets (of different sizes and image types) to show our attack is generally applicable. Future works may extend the evaluation to include more datasets (especially those used in cybercrime studies). Finally, our experiments on the defense part are limited (as it is not the main focus of this paper). We plan to systematically study countermeasure strategies in future works.

9 RELATED WORK

We discuss other related works to our paper. Those that are already described in Section 2 will not be repeated.

Adversarial Machine Learning. Our attack is related to adversarial machine learning attacks [11, 12, 17, 21, 43, 46–49, 51, 61, 67], with some noticeable differences. At the high level, adversarial examples construct the loss over *labels* (i.e., classes of samples) [23], whereas we construct the loss based on the perceptual hash distances between two specific images. Fundamentally, both attacks rely on running optimization over the defined losses. In this paper, we apply this idea to attack perceptual hashing applications and address application-specific challenges. First, perceptual hashing function is not differentiable and thus needs approximation; second, to speed up the black-box optimization, we introduce gray-scale initialization which exploits the gray-scale transformation step in perceptual hashing; third, we introduce various enhancements to attack real-world services.

We briefly discuss a few papers on adversarial machine learning that are related to our work. A recent study [70] illustrates various evasion attacks against perceptual ad-blocker (i.e., a classifier). The idea is to add noises to visual ads images so that a perceptual ad-blocker will classify them as "non-ad". Fawkes [60] applies adversarial machine learning to protect user privacy against facial recognition systems. The idea is to run *poisoning attacks* on the training process of facial recognition models, so that a user's face can no longer be correctly identified from the images. Our attack shares some similarity, in the sense that we can hide a face image from reverse image search. However, the methodology is fundamentally different, i.e., Fawkes requires *poisoning* the model training process. Another related work is the image-scaling attack [53, 82], which achieves adversarial evasion without directly manipulating a machine learning model. Instead, this attack manipulates the *preprocessing step* where images are re-scaled before they are sent to the machine learning pipeline. This attack is orthogonal to ours — it is possible we could combine image-scaling attack with our method to amplifies the attack impact against perceptual hashing.

Robustifying Perceptual Hashing. Researchers have worked to robustify perceptual hashing algorithms, but with a different threat model in mind. As we briefly discussed in Section 2, existing robust methods focus on threats against *image authentication*. Their goal is to accurately recognize malicious modifications (e.g., adding/removing an object in an image) by producing a different hash value [35, 42, 66, 68, 76, 81]. In the meantime, the robust method should produce the same hash value in the presence of normal image transformations such as scaling and rotation [39]. In our context (reverse image search), the goal is the opposite — a robust method should be more "tolerating" for modifications so that the search engine can successfully retrieve the source images (without introducing excessive false positives).

Recently, researchers also explored using deep neural networks (DNN) to augment existing hashing algorithms. The idea is to use DNNs to extract features that can better represent the neighborhood relationships [33, 75, 78]. Such DNNs can be unsupervised, but also can be supervised (by pairwise similarity labels) [10, 31, 32]. Unfortunately, DNN based hashing algorithms are extremely costly to train (at a large scale). In this paper, we focus on perceptual hashes that are widely used by current online services and security researchers. We defer the study of attacking DNN-based hashing algorithms to future work.

10 CONCLUSION

In this paper, we developed novel attack methods against perceptual hashing algorithms in the context of reverse image search. The attack challenges the validity of abuse detection and analysis tools that are built on perceptual hashing. We validated the impact of our attack using both controlled experiments and empirical tests on 4 real-world image search engines (TinEye, Google, Bing, and Yandex). Finally, we discussed (and briefly experimented with) a defense method that can be implemented on user clients.

ACKNOWLEDGMENT

We thank our shepherd Veelasha Moonsamy and the anonymous reviewers for their constructive comments and suggestions. This work was supported in part by NSF grants CNS-2030521 and CNS-2055233, and Amazon Research Award. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of any funding agencies.

REFERENCES

[1] Pushkal Agarwal, Kiran Garimella, Sagar Joglekar, Nishanth Sastry, and Gareth Tyson. 2020. Characterising User Content on a Multi-lingual Social Network. In *Proc. of ICWSM.*
[2] Pieter Agten, Wouter Joosen, Frank Piessens, and Nick Nikiforakis. 2015. Seven months' worth of mistakes: A longitudinal study of typosquatting abuse. In *Proc. of NDSS.*
[3] Anish Athalye, Logan Engstrom, Andrew Ilyas, and Kevin Kwok. 2018. Synthesizing Robust Adversarial Examples. In *Proc. of ICML.*
[4] Antonio Bianchi, Eric Gustafson, Yanick Fratantonio, Christopher Kruegel, and Giovanni Vigna. 2017. Exploitation and mitigation of authentication schemes based on device-public information. In *Proc. of AsiaCCS.*
[5] Microsoft Bing. 2021. Microsoft Bing Images. https://www.bing.com/images/trending.

[6] Kevin Borgolte, Christopher Kruegel, and Giovanni Vigna. 2015. Meerkat: Detecting website defacements through image-based object recognition. In *Proc. of USENIX Security*.

[7] Thomas Brunner, Frederik Diehl, Michael Truong Le, and Alois Knoll. 2019. Guessing Smart: Biased Sampling for Efficient Black-Box Adversarial Attacks. In *Proc. of ICCV*.

[8] Johannes Buchner. 2021. ImageHash. https://github.com/JohannesBuchner/imagehash.

[9] Elie Bursztein, Einat Clarke, Michelle DeLaune, David M. Elifff, Nick Hsu, Lindsey Olson, John Shehan, Madhukar Thakur, Kurt Thomas, and Travis Bright. 2019. Rethinking the Detection of Child Sexual Abuse Imagery on the Internet. In *Proc. of WWW*.

[10] Zhangjie Cao, Mingsheng Long, Jianmin Wang, and Philip Yu. 2017. HashNet: Deep Learning to Hash by Continuation. In *Proc. of ICCV*.

[11] Nicholas Carlini and David Wagner. 2017. Towards Evaluating the Robustness of Neural Networks. In *Proc. of IEEE SP*.

[12] Pin-Yu Chen, Huan Zhang, Yash Sharma, Jinfeng Yi, and Cho-Jui Hsieh. 2017. ZOO: Zeroth Order Optimization Based Black-Box Attacks to Deep Neural Networks without Training Substitute Models. In *Proc. of AISec*.

[13] Yuxuan Chen, Xuejing Yuan, Jiangshan Zhang, Yue Zhao, Shengzhi Zhang, Kai Chen, and XiaoFeng Wang. 2020. Devil's Whisper: A General Approach for Physical Adversarial Attacks against Commercial Black-box Speech Recognition Devices. In *Proc. of USENIX Security*.

[14] Jiri Fridrich and Miroslav Goljan. 2000. Robust Hash Functions for Digital Watermarking. *IEEE Int Conf Information Technology: Coding Computing* (2000).

[15] Robert Frischholz. 2021. Reverse Image Search — Searching People by Photos. https://facedetection.com/online-reverse-image-search/.

[16] Oana Goga, Giridhari Venkatadri, and Krishna P Gummadi. 2015. The doppelgänger bot attack: Exploring identity impersonation in online social networks. In *Proc. of IMC*.

[17] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014. Explaining and Harnessing Adversarial Examples. *arXiv preprint arXiv:1412.6572* (2014).

[18] Google. 2021. Google Image Search. https://www.google.com/imghp.

[19] Azhar Hadmi, William Puech, Brahim Said, and Abdellah Ouahman. 2013. A robust and secure perceptual hashing system based on a quantization step analysis. *Signal Processing: Image Communication* 28 (2013), 929–948.

[20] Chao-Yung Hsu, Chun-Shien Lu, and Soo-Chang Pei. 2009. Secure and robust SIFT. In *Proc. of Multimedia*.

[21] Weiwei Hu and Ying Tan. 2017. Generating Adversarial Malware Examples for Black-Box Attacks Based on GAN. *arXiv preprint arXiv:1702.05983* (2017).

[22] Andrew Ilyas, Logan Engstrom, Anish Athalye, and Jessy Lin. 2018. Black-box Adversarial Attacks with Limited Queries and Information. *arXiv preprint arXiv:1804.08598* (2018).

[23] Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Logan Engstrom, Brandon Tran, and Aleksander Madry. 2019. Adversarial Examples Are Not Bugs, They Are Features. In *Proc. of NeurIPS*.

[24] Infringement-Report. 2021. ImageRaider. https://infringement.report/api/raider-reverse-image-search/.

[25] Diane Kelly and Leif Azzopardi. 2015. How Many Results per Page? A Study of SERP Size, Search Behavior and User Experience. In *Proc. of SIGIR*.

[26] Amin Kharraz, William Robertson, and Engin Kirda. 2018. Surveylance: Automatically detecting online survey scams. In *Proc. of IEEE SP*.

[27] Seon Joo Kim. 2021. Real and Fake Face Detection: Discriminate Real and Fake Face Images. https://www.kaggle.com/ciplab/real-and-fake-face-detection.

[28] Panagiotis Kintis, Najmeh Miramirkhani, Charles Lever, Yizheng Chen, Rosa Romero-Gómez, Nikolaos Pitropakis, Nick Nikiforakis, and Manos Antonakakis. 2017. Hiding in plain sight: A longitudinal study of combosquatting abuse. In *Proc. of CCS*.

[29] Evan Klinger and David Starkweather. 2021. pHash: The open source perceptual hash library. https://www.phash.org/docs/.

[30] Ching-Yung Lin and Shih-Fu Chang. 2001. A robust image authentication system distinguishing JPEG compression from malicious manipulation. *IEEE Transactions on Circuits and Systems for Video Technology* 11 (2001).

[31] Haomiao Liu, Ruiping Wang, Shiguang Shan, and Xilin Chen. 2016. Deep Supervised Hashing for Fast Image Retrieval. In *Proc. of CVPR*.

[32] Wei Liu, Jun Wang, Rongrong Ji, Yu-Gang Jiang, and S. Chang. 2012. Supervised Hashing with Kernels. In *Proc. of CVPR*.

[33] Wei Liu, Jun Wang, Sanjiv Kumar, and Shih-Fu Chang. 2011. Hashing with Graphs. In *Proc. of ICML*.

[34] Yanpei Liu, Xinyun Chen, Chang Liu, and Dawn Song. 2017. Delving into Transferable Adversarial Examples and Black-box Attacks. In *Proc. of ICLR*.

[35] David G. Lowe. 2004. Distinctive Image Features from Scale-Invariant Keypoints. *Int. J. Comput. Vision* 60, 2 (2004).

[36] Commons Machinery. 2021. Blockhash. http://blockhash.io/.

[37] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2018. Towards Deep Learning Models Resistant to Adversarial Attacks. In *Proc. of ICLR*.

[38] Philipe Melo, Johnnatan Messias, Gustavo Resende, Kiran Garimella, Jussara Almeida, and Fabrício Benevenuto. 2019. Whatsapp monitor: A fact-checking system for whatsapp. In *Proc. of ICWSM*.

[39] M. Kıvanç Mıhçak and Ramarathnam Venkatesan. 2002. New Iterative Geometric Methods for Robust Perceptual Image Hashing. In *Proc. of ACM Workshop on Security and Privacy in Digital Rights Management*.

[40] Najmeh Miramirkhani, Oleksii Starov, and Nick Nikiforakis. 2016. Dial one for scam: Analyzing and detecting technical support scams. In *Proc. of NDSS*.

[41] Alexandros Mittos, Savvas Zannettou, Jeremy Blackburn, and Emiliano De Cristofaro. 2020. "And We Will Fight for Our Race!" A Measurement Study of Genetic Testing Conversations on Reddit and 4chan. In *Proc. of ICWSM*.

[42] Vishal Monga and Brian L. Evans. 2004. Robust perceptual image hashing using feature points. In *Proc. of ICIP*.

[43] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. 2016. DeepFool: A Simple and Accurate Method to Fool Deep Neural Networks. In *Proc. of CVPR*.

[44] Nick Nikiforakis, Federico Maggi, Gianluca Stringhini, M Zubair Rafique, Wouter Joosen, Christopher Kruegel, Frank Piessens, Giovanni Vigna, and Stefano Zanero. 2014. Stranger danger: exploring the ecosystem of ad-based url shortening services. In *Proc. of WWW*.

[45] Adam Novozámský, Babak Mahdian, and Stanislav Saic. 2020. IMD2020: A Large-Scale Annotated Dataset Tailored for Detecting Manipulated Images. In *Proc. of IEEE Winter Applications of Computer Vision Workshops*.

[46] Thomas Kobber Panum, Kaspar Hageman, René Rydhof Hansen, and Jens Myrup Pedersen. 2020. Towards Adversarial Phishing Detection. In *Proc. of CSET*.

[47] Nicolas Papernot, Patrick McDaniel, and Ian Goodfellow. 2016. Transferability in Machine Learning: from Phenomena to Black-Box Attacks using Adversarial Samples. *arXiv preprint arXiv:1605.07277* (2016).

[48] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. 2017. Practical Black-Box Attacks against Machine Learning. In *Proc. of AsiaCCS*.

[49] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. 2016. The limitations of deep learning in adversarial settings. In *Proc. of IEEE SP*.

[50] Sergio Pastrana, Alice Hutchings, Daniel Thomas, and Juan Tapiador. 2019. Measuring EWhoring. In *Proc. of IMC*.

[51] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. 2017. DeepXplore: Automated Whitebox Testing of Deep Learning Systems. In *Proc. of SOSP*.

[52] Pixsy. 2021. Pixsy. https://www.pixsy.com/.

[53] Erwin Quiring, David Klein, Daniel Arp, Martin Johns, and Konrad Rieck. 2020. Adversarial Preprocessing: Understanding and Preventing Image-Scaling Attacks in Machine Learning. In *Proc. of USENIX Security*.

[54] M. Zubair Rafique, Tom Van Goethem, Wouter Joosen, Christophe Huygens, and Nick Nikiforakis. 2017. It's Free for a Reason: Exploring the Ecosystem of Free Live Streaming Services. In *Proc. of NDSS*.

[55] Jathushan Rajasegaran, Naveen Karunanayake, Ashanie Gunathillake, Suranga Seneviratne, and Guillaume Jourjon. 2019. A multi-modal neural embeddings approach for detecting mobile counterfeit apps. In *Proc. of WWW*.

[56] Julio C.S. Reis, Philipe Melo, Kiran Garimella, Jussara M Almeida, Dean Eckles, and Fabrício Benevenuto. 2020. A Dataset of Fact-Checked Images Shared on WhatsApp During the Brazilian and Indian Elections. In *Proc. of ICWSM*.

[57] Gustavo Resende, Philipe Melo, Hugo Sousa, Johnnatan Messias, Marisa Vasconcelos, Jussara Almeida, and Fabrício Benevenuto. 2019. (Mis) Information Dissemination in WhatsApp: Gathering, Analyzing and Countermeasures. In *Proc. of WWW*.

[58] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. 2015. ImageNet Large Scale Visual Recognition Challenge. In *Proc. of IJCV*.

[59] Marc Schneider and Shih-Fu Chang. 1996. A robust content based digital signature for image authentication. In *Proc. of ICIP*.

[60] Shawn Shan, Emily Wenger, Jiayun Zhang, Huiying Li, Haitao Zheng, and Ben Zhao. 2020. Fawkes: Protecting Personal Privacy against Unauthorized Deep Learning Models. In *Proc. of USENIX Security*.

[61] Mahmood Sharif, Sruti Bhagavatula, Lujo Bauer, and Michael K. Reiter. 2016. Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition. In *Proc. of CCS*.

[62] Shangcheng Shi, Xianbo Wang, and Wing Cheong Lau. 2019. MoSSOT: An Automated Blackbox Tester for Single Sign-On Vulnerabilities in Mobile Applications. In *Proc. of AsiaCCS*.

[63] Rachee Singh, Rishab Nithyanand, Sadia Afroz, Paul Pearce, Michael Carl Tschantz, Phillipa Gill, and Vern Paxson. 2017. Characterizing the nature and dynamics of Tor exit blocking. In *Proc. of USENIX Security*.

[64] Kate Starbird, Ahmer Arif, and Tom Wilson. 2019. Disinformation as Collaborative Work: Surfacing the Participatory Nature of Strategic Information Operations. *CSCW* 3 (2019).

[65] Fnu Suya, Jianfeng Chi, David Evans, and Yuan Tian. 2020. Hybrid Batch Attacks: Finding Black-box Adversarial Examples with Limited Queries. In *Proc. of USENIX*

Security.
[66] Ashiwin Swaminathan, Yinian Mao, and Min Wu. 2006. Robust and secure image hashing. *IEEE Transactions on Information Forensics and Security* 1, 2 (2006), 215–230.
[67] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2013. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199* (2013).
[68] Zhenjun Tang, Xianquan Zhang, and Shichao Zhang. 2014. Robust Perceptual Image Hashing Based on Ring Partition and NMF. *ITKDE* 26, 3 (2014), 711–724.
[69] TinEye. 2021. TinEye. https://tineye.com/.
[70] Florian Tramèr, Pascal Dupré, Gili Rusak, Giancarlo Pellegrino, and Dan Boneh. 2019. AdVersarial: Perceptual Ad Blocking meets Adversarial Machine Learning. In *Proc. of CCS*.
[71] Phani Vadrevu and Roberto Perdisci. 2019. What You See is NOT What You Get: Discovering and Tracking Social Engineering Attack Campaigns. In *Proc. of IMC*.
[72] Tom Van Goethem, Najmeh Miramirkhani, Wouter Joosen, and Nick Nikiforakis. 2019. Purchased Fame: Exploring the Ecosystem of Private Blog Networks. In *Proc. of AsiaCCS*.
[73] Vinny Vella. 2020. Video of Delaware County poll workers filling out ballots was manipulated and lacked context, officials say. Inquirer. (6 November 2020).
[74] Ramarathnam Venkatesan, S.-M Koon, Mariusz Jakubowski, and Pierre Moulin. 2000. Robust image hashing. In *Proc. of ICIP*.
[75] J. Wang, S. Kumar, and S. Chang. 2012. Semi-Supervised Hashing for Large-Scale Search. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2012).
[76] Xiaofeng Wang, Kemu Pang, Xiaorui Zhou, Yang Zhou, Lu Li, and Jianru Xue. 2015. A Visual Model-Based Perceptual Image Hash for Content Authentication. *IEEE Transactions on Information Forensics and Security* 10, 7 (2015), 1336–1349.
[77] Yuping Wang, Fatemeh Tahmasbi, Jeremy Blackburn, Barry Bradlyn, Emiliano De Cristofaro, David Magerman, Savvas Zannettou, and Gianluca Stringhini. 2021. Understanding the Use of Fauxtography on Social Media. In *Proc. of ICWSM*.
[78] Yair Weiss, Antonio Torralba, and Rob Fergus. 2009. Spectral Hashing. In *Proc. of NeurIPS*.
[79] Li Weng and Bart Preneel. 2007. Attacking Some Perceptual Image Hash Algorithms. In *Proc. of ICME*.
[80] Li Weng and Bart Preneel. 2011. A Secure Perceptual Hash Algorithm for Image Content Authentication. In *Proc. of CMS*.
[81] Zhong Wu, Qifa Ke, Michael Isard, and Jian Sun. 2009. Bundling Features for Large Scale Partial-Duplicate Web Image Search. In *Proc. of CVPR*.
[82] Qixue Xiao, Yufei Chen, Chao Shen, Yu Chen, and Kang Li. 2019. Seeing is Not Believing: Camouflage Attacks on Image Scaling Algorithms. In *Proc. of USENIX Security*.
[83] Yandex. 2021. Yandex Image Search. https://yandex.com/images/.
[84] Bian Yang, Fan Gu, and Xiamu Niu. 2006. Block Mean Value Based Image Perceptual Hashing. In *Proc. of IIHMSP*.
[85] Kan Yuan, Di Tang, Xiaojing Liao, XiaoFeng Wang, Xuan Feng, Yi Chen, Menghan Sun, Haoran Lu, and Kehuan Zhang. 2019. Stealthy porn: Understanding real-world adversarial images for illicit online promotion. In *Proc. of IEEE SP*.
[86] Savvas Zannettou, Tristan Caulfield, Jeremy Blackburn, Emiliano De Cristofaro, Michael Sirivianos, Gianluca Stringhini, and Guillermo Suarez-Tangil. 2018. On the Origins of Memes by Means of Fringe Web Communities. In *Proc. of IMC*.
[87] Savvas Zannettou, Tristan Caulfield, Barry Bradlyn, Emiliano De Cristofaro, Gianluca Stringhini, and Jeremy Blackburn. 2020. Characterizing the Use of Images in State-Sponsored Information Warfare Operations by Russian Trolls on Twitter. In *Proc. of ICWSM*.
[88] Christoph Zauner. 2010. Implementation and Benchmarking of Perceptual Image Hash Functions. Master's thesis, Upper Austria University of Applied Sciences, Hagenberg Campus. (2010).
[89] Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. 2018. The Unreasonable Effectiveness of Deep Features as a Perceptual Metric. In *Proc. of CVPR*.

A PERCEPTUAL HASHING

In this section, we provide more details for the perceptual hashing algorithms used in our paper.

pHash. pHash first down-scales the image to $32 \times 32 = 1024$ pixels, and converts it into the grayscale. Then it applies Discrete Cosine Transform (DCT) to the image and computes a DCT hash using the low DCT coefficients (i.e., the top 8×8=64 low frequency part). It then calculates the median of the DCT coefficients. The binary hash string is generated depending on if each DCT frequency is below or above the median. The final hash string is constructed by converting the binary hash string into a hex string. pHash already considers robustness in its design as it describes the relative frequency with respect to the median. It is designed to tolerate image modifications as long as the main structure of the image stays the same.

Blockhash. Blockhash slices the RGB image into blocks (i.e., the default number is 16×16=256 blocks). If the image is dividable by the required number of blocks, it will trigger a quick version for hash computation. Otherwise, a slow version (with additional processing) will be triggered. We pre-process our images such that the images are dividable by the number of blocks. After the image is divided into 256 blocks, the algorithm then assigns a value to each block (which is the sum of all the pixels' RGB values in this block). After that, the algorithm divides these 256 blocks into 4 groups (64 blocks per group). For each group, it computes the median of the block values. To compute the binary string (256-bit), each bit is calculated based on one block. The bit value is based on whether this block's value is above or below the median of the block's corresponding group. Finally, the binary string is converted to a hex string. Unlike pHash which calculates the hash string based on the frequency domain information, Blockhash relies on the color domain information of different blocks.

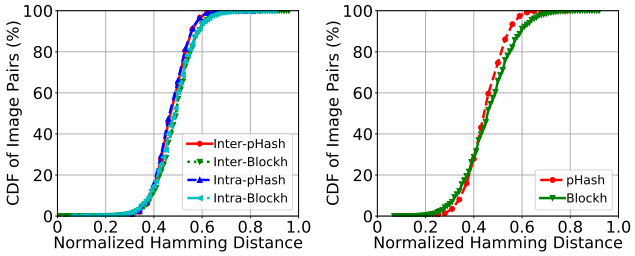
aHash. aHash is short for Average Hash. It reduces the image into a fixed smaller size (i.e., 8×8=64 pixels) and then converts the image into grayscale. Then it calculates the *mean* value of the 64 pixel/color values. The binary hash string is calculated based on whether each pixel value is above or below the mean. Finally, aHash converts the binary string into a hex string. aHash is fast but it easily generates false positives as it tolerates more differences.

dHash. dHash is short for Difference Hash. dHash computes the hash string based on the differences between adjacent columns. dHash down-scales the image into a smaller dimension (i.e., 9×8=72 pixels) and converts it into grayscale. Then dHash calculates the binary string based on adjacent pixels. The 9 pixels in each row will generate 8 values for the differences between the adjacent pixels (64 bits in total combining the 8 rows). dHash is comparable to aHash in terms of efficiency, but dHash is more *precise* in image matching.

wHash. wHash stands for Wavelet Hash. wHash first re-scales the image to a smaller size (i.e., 8×8=64 pixels) and converts it into grayscale. Then wHash uses Discrete Wavelet Transformation (haar transformation to be more precise) to extract features from the frequency domain to compute the hash string. Note that wHash excludes the lowest frequency part. The lowest frequency consists of only one data point that represents the contrast of the image (which is not useful).

B HASH DISTANCE OF IMAGES

Below, we provide some reference points for hashing distance. Given an image dataset, we examine the normalized Hamming distance between the hash codes of a random pair of images. The results are shown in Figure 8. For ImageNet, we randomly sample 100 classes and then randomly sample 100 images from each class. We examine the distance of *all* image pairs. We break down the results for image pairs from the same classes and those from two different classes. We find that the inter-class distance and intra-class



(a) ImageNet **(b) Face**

Figure 8: Normalized Hamming distance of image pairs.

distance does not have significant differences. Given random image pairs (i.e., two different images), using a distance threshold of 0.2 can safely separate them apart. For the Face dataset, we exhaustively compute the distance of *all* image pairs. Similarly, a distance of 0.2 can separate different images in the pair. We did not show the results for the IMD dataset here since it has a different setup: this dataset contains pairs of original and manipulated images. Further analysis of IMD is in Section 5.6.

C EXAMPLES FOR IMD DATASET

Certain image pairs in the IMD Dataset already have a large pHash distance between the manipulated image and the original image (e.g., >0.2). Figure 9 shows an example where the pHash distance of the pair is 0.56. Such manipulated images often involve significant modifications (e.g., the original image is no longer the main body of the manipulated image).

Clean **Manipulated**

Figure 9: IMD image pairs with a large pHash distance (0.56).

D CLIENT-SIDE DEFENSE EXPERIMENT

In this section, we perform a preliminary experiment for our user-side defense idea. The purpose of this experiment is to provide some supporting evidence for our discussions regarding countermeasure strategies. Since the experiments are only preliminary, we do not claim this as a main contribution of the paper.

Given an input image $\boldsymbol{x}$, our idea is to build a browser extension that automatically generates multiple search queries to search a set of similar-looking images $\{\boldsymbol{x'}\}$. We hope some of the images in $\{\boldsymbol{x'}\}$ can destroy the adversarial noises in the original input, and help to return the true matching images.

Generating $\{\boldsymbol{x'}\}$. As a simple experiment, we design 8 transformations to generate $\{\boldsymbol{x'}\}$. Each time when the transformation is used by a particular user, we pick random parameters for the transformation such that attackers cannot pre-optimize the adversarial noise that would universally work for *all the users on the Internet*. As a proof-of-concept experiment, the transformations we used include rotation, scaling, and denoising image filters.

Given an input $\boldsymbol{x}$, we apply these transformations to generate 8 images. More specifically, we produce 2 images using rotation: one rotation angle is randomly chosen from $[0.01{\cdot}2\pi, 0.1{\cdot}2\pi]$ and the other rotation angle is randomly chosen from $[-0.1{\cdot}2\pi, 0)$. Then we produce 4 more images from scaling: two scaling factors r_w are randomly chosen from [1.01,1.21) to scale the image width r_w times; Two scaling factors r_h are randomly chosen from [1.1,1.3) to scale the image height r_h times. Then 2 more images are produced by combining height-scaling with a random image filter (selected from BLUR, SMOOTH-MORE, EDGE_ENHANCE_MORE). In this way, 8 randomly transformed images are generated to form $\{\boldsymbol{x'}\}$.

Performing Searches. For this experiment, we take all the attack images from Section 6.3. These are the most aggressive attack images generated by the AoT3-pHash attack. For each attack image (treated as a $\boldsymbol{x}$), we generated 8 transformed images $\{\boldsymbol{x'}\}$. Then we use each transformed image to perform a Google reverse search and obtain the returned results.

Fusing Search Results. There are many ways to fuse the search results so that the true matching images can be ranked to the top. Again, as a proof-of-concept, we design a simple fusion method. Here, we need to merge 8 ranked lists into 1 ranked list. For each entry, we rank it based on two factors (1) how often this entry appears across the 8 lists; and (2) this entry's original ranking in its own list.

Suppose an entry $e_{i,j}$ is the i_{th} ranked entry on the list j (j = 1, ..., 8). We compute a tuple for this entry as $(count(e_{i,j}), i)$. Here, the function $count(e_{i,j})$ counts how many times this entry appears across the 8 lists (based on the entry URL or the image file). To create one ranked list, we collect all the entries (from 8 lists) and their corresponding tuples. We first rank the entries based on the $count(e_{i,j})$ value in the tuple. For entries that share the same $count$, we then refer to the original ranking i in their original list to rank them. In this way, all the entries are ranked in a single list.

Results and Observations. We perform the above experiment on Google for all the attack images generated in Section 6.3. For each image, we produce one ranked list using the fusion method described above. Then we manually inspect the top-50 results of the fused list and calculate the top-1 and top-10 hit rates and false positives as before. Comparing with the results in Table 9 (no defense), we find that the defense method indeed has a better performance. The defense method has improved the top-1 hit rate from 36% to 70%. The top-10 hit rate is also improved from 36% to 76%. In the meantime, the average false positive rate of the defense method is 39.1%, which is also much lower than the original 66% without defense. As a preliminary experiment, the result shows the initial promise for the proposed idea. We believe more work should be done to improve the "naive" methods for generating $\{\boldsymbol{x'}\}$ and fusing the search results, to further improve the defense performance.

E FAILED CASE ANALYSIS

As shown in Section 6, our attacks on real-world search engines still have failed cases (especially on Google). We took the failed

images and manually inspected their search results. We did not find clear patterns among these failed cases in comparison with the successful ones. For example, the number of indexed copies of the query image is not a clear predictor of success. In other words, the attack is not necessarily more successful on query images that are heavily/rarely indexed by the target search engine. In addition, we do not think the attack image's quality is a strong predictor of success, i.e., attack images with more visible noises are not necessarily more successful. Based on manual inspections, we cannot conclusively determine the root causes (given we do not have any knowledge of Google's internal searching algorithms and potential robustification methods).