Using Perceptual Hashing for Targeted Content Scanning

Leon Twenning, Harald Baier, and Thomas Göbel

Bundeswehr University, Munich, Germany
thomas.goebel@unibw.de

Abstract. The Internet is increasingly used to disseminate unethical and illegal content. A grave concern is child sexual abuse material that is often disseminated via end-to-end-encrypted channels. Such encryption defeats network- and server-based scanning measures used by law enforcement. A trade-off is to enable confidential communications channels for users and scanning opportunities for law enforcement by employing perceptual-hashing-based targeted content scanning on user devices. This has generated intense discussions between policymakers, privacy advocates and child protection organizations.
This chapter summarizes the current state of research in perceptual-hashing-based targeted content scanning with a focus on classical metrics such as false positives, false negatives and privacy aspects. Insights are provided into the most relevant perceptual hashing methods and an attack taxonomy for perceptual-hashing-based targeted content scanning is presented. The complexity in generating false negatives is evaluated and the feasibility of evading perceptual-hashing-based targeted content scanning is demonstrated.

Keywords: Perceptual Hashing · Targeted Content Scanning · Child Sexual Abuse Material

1 Introduction

Digital communications channels such as messenger services, discussion boards and social networks are increasingly used for nefarious activities, a trend that appears to have increased during the COVID-19 pandemic [28]. The communications channels are low-cost, highly-scalable and private, and perpetrators leverage them to disseminate illegal, abusive and violent content. The content is often in the form of images and videos that contain hate speech, disinformation, terrorist propaganda and more.

A grave concern is the dissemination of child sexual abuse material (CSAM). According to the German Federal Criminal Police Office [9], the overall numbers of crimes related to child sexual abuse were nearly seven times higher in 2021 compared with 2016. In 2021, the U.S. National Center for Missing and Exploited Children (NCMEC) [20] collected nearly 30 million reports of CSAM from large technology companies such as Meta and Google.

Published by Springer Nature Switzerland AG 2023
G. Peterson and S. Shenoi (Eds.): DigitalForensics 2023, IFIP AICT 687, pp. 125–142, 2023.
https://doi.org/10.1007/978-3-031-42991-0_7

Several companies have cracked down on the dissemination of illegal content via their online services. While the technical details of their countermeasures are not released, it is clear that image recognition technologies are employed to scan content. Major companies that conduct scanning include Meta [8], Microsoft [19], Apple [2] and Google [26].

Perpetrators are increasingly disseminating illegal content such as CSAM via end-to-end-encrypted channels. Such encryption defeats network- and server-based scanning measures used by law enforcement. A trade-off is to enable confidential communications channels for users and scanning opportunities for law enforcement by employing perceptual-hashing-based targeted content scanning on user devices.

This chapter summarizes the current state of research in perceptual-hashing-based targeted content scanning with a focus on classical metrics such as false positives, false negatives and privacy aspects. Insights are provided into the most relevant perceptual hashing methods and an attack taxonomy for perceptual-hashing-based targeted content scanning is presented. The complexity in generating false negatives is evaluated and the feasibility of evading perceptual-hashing-based targeted content scanning is demonstrated.

2 Related Work

NIST Special Publication 800-168 [7] defines approximate matching as "a generic term describing any technique designed to identify similarities between two digital artifacts." The publication also categorizes a perceptual hash as a semantic hash used primarily for bytewise approximate matching.

Academic research on perceptual hashing is relatively sparse. However, several open-source perceptual hashes have been developed, including bHash [29], aHash [14], dHash [15], pHash [13, 30], wHash [21] and PDQ [17, 18]. Of these perceptual hashes, only pHash and PDQ come with detailed descriptions. In 2010, Zauner [30] provided an extensive description and evaluation of pHash; despite its age, pHash is still relevant today. Meta, which developed PDQ, has released a technical paper [18] covering the inner workings of the hash and a limited evaluation [17].

Closed-source perceptual hashes have also been developed, the most commonly used being PhotoDNA [19] and NeuralHash [2]. While no official document related to PhotoDNA is available, Apple, the NeuralHash developer, has released limited technical information about the perceptual hash [2, 3].

Kulshrestha and Mayer [16] are the only academic researchers to propose a perceptual-hashing-based targeted content scanning system that is designed in a privacy-preserving manner. Their proposal seeks to enforce privacy using cryptographic methods, specifically, private exact membership computation (PEMC) and private approximate membership computation (PAMC). They do not judge whether or not it is ethical to use such a system nor do they take positions on what targeted content should be scanned and the effectiveness of scanning.

Several academic researchers have focused on attacking perceptual hashes. Struppek et al. [25] presented four attacks on NeuralHash that enable detection evasion, hash collision and information extraction. Evaluations of the attacks using a dataset of 10,000 images demonstrate that they are functional and realistic. Importantly, Struppek and colleagues are the only researchers to release the actual code used to evaluate attacks. Jain et al. [11] presented detection evasion attacks on the pHash, aHash, dHash and PDQ perceptual hashes. Their detailed evaluation employed images from the same dataset used by Struppek et al. [25], but their sample size was much larger.

3 Perceptual Hashing

This section explains the general concept of perceptual hashing and presents details about two prominent hash functions.

3.1 Perceptual Hashing Details

Perceptual hashing is an approximate matching technique for comparing the similarity of objects. In this work, the focus is on the perceptual similarity of pictures (images). Perceptual similarity means that a human presented with two images would judge them as being the same or similar. Because images are often modified slightly during their use (e.g., compressed to reduce bandwidth and storage before being uploaded to a digital service), they cannot be identified by cryptographic hashes or other exact matching techniques. In such scenarios, perceptual hashing can be used effectively for approximate matching [16]. Perceptual hashing operates at the semantic level of images, interpreting the internal data structures of image files and using perceptive features of the images for matching [7].

As an approximate matching technique, perceptual hashing has two basic components. One is a similarity digest based on object features and the other is a similarity function that compares similarity digests and computes a similarity score [7].

A feature is the most basic component of an object that is compared and comparing two features yields a binary result [7]. In the case of perceptual hashes, the features should map to data representing the human perceptions of images.

Following the notation of Struppek et al. [25], a perceptual hash function H is given by:

$$H : \mathbb{R}^{h \times w \times c} \rightarrow \{0,1\}^k$$

where the three-dimensional array input to function H contains image height h, width w and color channel c data, and the output of function H is a k-bit binary hash. Note that k is a fixed value chosen for a concrete perceptual hash to ensure that hashes produced by the hash function are of equal length. A hash computation involves two parts. First, image features are extracted and stored in an intermediary format (e.g., matrix or vector). Next, the intermediary format is mapped to a k-bit hash.

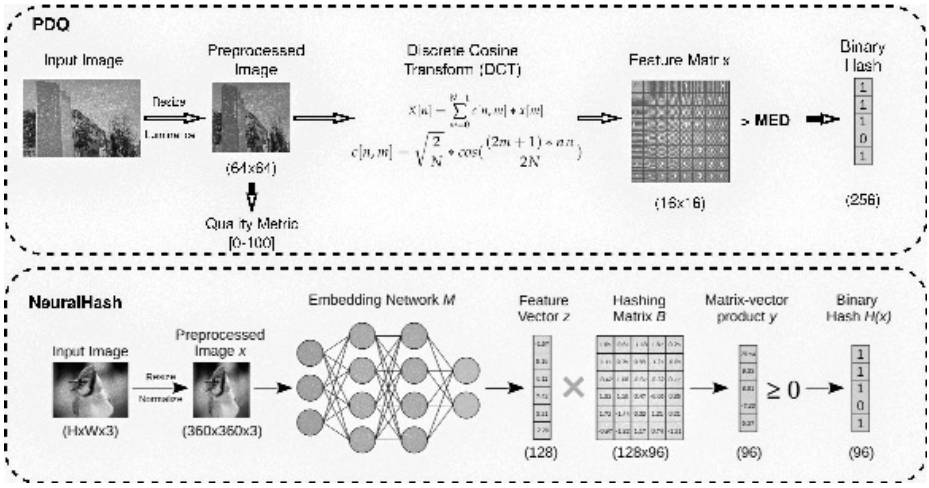

Fig. 1. Overview of PDQ and NeuralHash.

A similarity function $D(h_1, h_2)$ is used to compare two perceptual hashes h_1 and h_2. It produces a similarity score between 0 and 100. The higher the score of images with hashes h_1 and h_2, the more perceptually similar the images. Hamming distance is often used as the basic comparison function. Two images are determined to be perceptually the same, if their similarity score exceeds a preset threshold [30].

3.2 Perceptual Hash Functions

This section describes two prominent perceptual hash functions, PDQ from Meta and NeuralHash from Apple. Figure 1 provides an overview of the main steps in the two functions. Note that the NeuralHash image in Figure 1 is taken from [25].

Although certain differences exist between the two perceptual hash functions, it is possible to generalize the process. Specifically, the input image data is normalized for further processing and the image details are reduced by rescaling and removing color. Next, features representing the perceptions of images are extracted. The features are mapped to a binary hash. While image normalization and the final reduction step have big influences on perceptual hash quality, the main distinction between the two implementations is the feature extraction process. PDQ relies on digital image processing using the discrete cosine transform whereas NeuralHash uses a neural network.

PDQ PDQ was designed to detect abusive imagery in large-scale contexts in social networks by comparing digital perceptual hashes [17]. While PDQ is claimed to be built from the ground up, it is very similar to the discrete-cosine-transform-based pHash [13].

Figure 1 (top) shows the steps involved in generating a PDQ hash of an image [18]. First, the image is transformed to its luminance representation. To further reduce the image details and render the hash computations less resource intensive, the image is resized to a 64×64 pixel square. The interpolation method employed ensures that each pixel contributes to a down-sampled pixel.

Next, the most perceptually relevant portions of the 64×64 image are extracted. This is accomplished by using the discrete cosine transform to convert the image to a frequency representation and portions of the image that do not contribute much to human perception are discarded. The 64×64 matrix output of the discrete cosine transform is thus reduced to a 16×16 submatrix.

Finally, to generate the hash, the median value of the 16×16 submatrix is computed. Each value in the submatrix is compared against the median to create the 256-bit PDQ hash. Specifically, when a submatrix value is larger than the median, a one is placed in the hash; otherwise, a zero is placed in the hash.

NeuralHash NeuralHash is intended to be a part of a CSAM protection system deployed in Apple's ecosystem of clients and cloud services. Apple released limited information about NeuralHash in a technology description [2] and threat model review [3] before its planned release with iOS 15 in late 2021. Additional details about NeuralHash are described in [25] based on community efforts to extract and reverse engineer its algorithm.

Figure 1 (bottom) shows the steps involved in generating a NeuralHash of an image. First, the input image is preprocessed. This involves transforming the image to the RGB color model, resizing it to a 360×360 pixel square and normalizing the RGB pixel values to $[-1, 1]$ [25]. Next, a neural network is used to extract image features in the form of an array with floating point values. The values serve as descriptors or features of the image [2]. The neural network was previously trained in a self-supervised manner using an unknown dataset to generate close descriptors for images that are perceptually similar. The perceptual similarity is based on angular distance and cosine similarity.

Following this, the array of descriptors is compressed into a much smaller 96-bit hash that preserves similarity using a locality-sensitive hash [2]. Note that NeuralHash was not trained to detect CSAM *per se*, but rather to compare images that are perceptual similar. Therefore, despite the fact that it uses a neural network for feature extraction, NeuralHash is characterized as a perceptual hash function.

4 Targeted Content Scanning

A perceptual-hashing-based targeted content scanning (PHTCS) system comprises three core components, a perceptual hash function, the targeted content database and processes and policies that determine which images are targeted during scanning and how the recognized targeted content is moderated.

The main goal of PHTCS is to find and moderate occurrences of targeted content in a digital service or communications channel. Perceptual hashes are

used to scan a flow of images and recognize the images with targeted content [1, 16].

A targeted content database comprising a reference list of perceptual hashes is employed. The database can be viewed as a blacklist. Content subject to PHTCS is examined using perceptual hashing. This enables blacklisted images as well as images that are perceptually similar to be identified. A clear limitation of the blacklisting employed by PHTCS is that only known targeted content can be recognized [1].

Finally, processes and policies must be defined. A PHTCS system is enrolled in a digital service, which feeds it images that it scans to recognize targeted content. Therefore, PHTCS is typically specified for a single service or single service provider that enrolls it in its services. While software is required to implement the image flow, more interesting are the processes and policies that specify how scanning is conducted and how images that match the targeted content are moderated [16].

In general, there needs to be a process that regulates the triggers that cause images to be scanned. For example, scanning could be triggered by a user uploading an image to a social network. Before the image is processed further and displayed in the social network, it would be input to the PHTCS system. This leads to a second process that regulates the actions of the social network if an uploaded image is deemed to have targeted content.

Some of the PHTCS actors have already been mentioned. One actor is the service provider that has enrolled PHTCS in one or more of its services. Another actor is a user of the digital service that has enrolled PHTCS. Aside from triggering scanning due to the presence of targeted content, the user has no active role in PHTCS [1, 16]. Another actor that may not be immediately obvious is a trusted party, which is responsible for creating and curating the targeted content database. In the case of CSAM scanning, child safety organizations, such as the NCMEC in the United States, would curate databases of perceptual hashes and share them with partners. This actor is called a trusted party because the service provider has to trust the actor to include only hashes of targeted content in the database. Of course, this would not be verifiable by the service provider or users because a perceptual hash is a one-way function and the actual targeted content would not be shared [16]. The last PHTCS actor is law enforcement that comes into play when illegal content is detected. The content may be reported by the service provider to the responsible law enforcement agency on a voluntary or mandatory basis [1].

5 Attacking PHTCS

PHTCS relies on perceptual hashes to recognize images that are defined as targeted content. Users intending to circumvent or abuse a PHTCS system would be interested in launching attacks on PHTCS. Some PHTCS systems may implement protection mechanisms against attacks. For example, human review of

each matched image can prevent misinterpretations of benign images as targeted content. However, this additional step is expensive and is difficult to scale.

As a result, companies would design their PHTCS systems to have very low (natural) false positive rates (see, e.g., [6]). Errors or adversarial methods that prevent PHTCS from detecting harmful images as targeted content cannot be prevented by human review. Users that manipulate their harmful images to evade detection could significantly reduce PHTCS effectiveness.

Perceptual hashes are more likely to be attacked successfully if they are used in client-side PHTCS. This is because the attacker could gain direct access to key PHTCS components, namely, the targeted content database and the device that performs the scanning. In server-side scanning, attacker access would be much more limited and the perceptual hashes would often be kept secret (e.g., with PhotoDNA [1]).

Based on research described in [22, 25], a taxonomy of attacks on perceptual hashes in the context of targeted content scanning is created. The taxonomy differentiates attacks into three categories, detection evasion, hash collision and data leakage. The following sections describe the attacks and how attackers can use them to combat PHTCS. Additionally, practical examples of attacks on perceptual hashes and their ability to hinder PHTCS are described.

5.1 Detection Evasion

A detection evasion attack seeks to prevent an image containing targeted content from being matched by a PHTCS system. This is accomplished by transforming an image X (hash $H(X)$) with targeted content to a new image X' (hash $H(X')$ that is perceptually similar to X but $H(X') \neq H(X)$. If the PHTCS system to be bypassed employs a threshold T for matching hashes, then the similarity $D(H(X), H(X')) > T$ must hold [10, 11]. For the attack to be effective, changes to the image should have minimal impacts on its perception, but its perceptual hash value would be altered significantly [11]. To carry out the attack, an attacker must have access to the image X that must evade detection and should be able to compute the perceptual hashes $H(X)$ and $H(X')$ [11].

Detection evasion attacks would enable users to store, view and disseminate images with targeted content on services and devices with enrolled PHTCS systems. A successful attack would impact PHTCS effectiveness significantly. The impact on effectiveness is evaluated using three metrics:

- **Success Rate:** This metric assesses how reliably an attack works on images.
- **Perceptual Similarity:** This metric assesses the similarity of an evading image compared with an image with targeted content.
- **Modification Effort:** This metric considers the effort, in terms of time, involved in modifying an image with targeted content. Simple image modifications such as rotating or resizing can be used, as well as more complex modifications such as editing pixels in special positions based on the perceptual hashes [25].

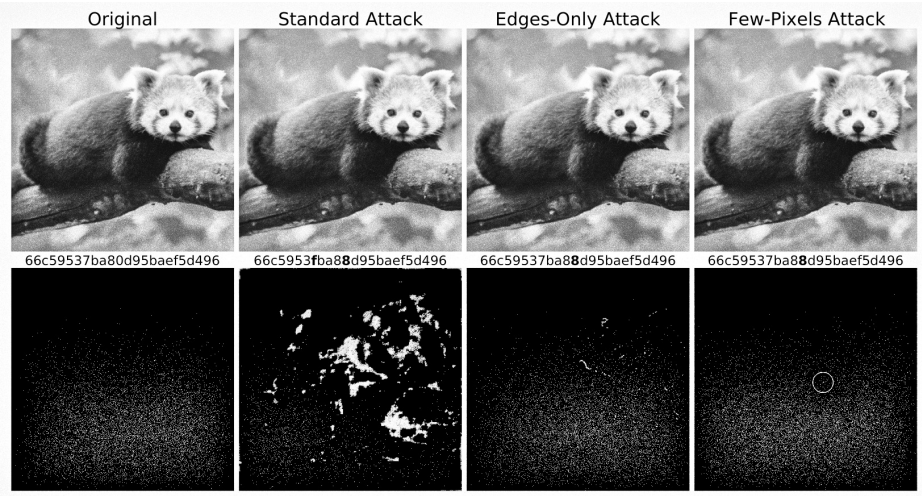

Fig. 2. Detection evasion attacks on NeuralHash (from [25]).

NeuralHash Attacks Struppek et al. [25] describe detection evasion attacks that employ gradient-based modifications of images. The attacks introduce specific perturbations to an original image X with targeted content. The perturbations are obtained by one or more optimization steps using a generic property of the neural network used by NeuralHash to extract image features. Specifically, that hash computations are differentiable and gradient descent can be used to find hash collisions [5].

Struppek et al. [25] demonstrated three levels of the attack that depend on the perturbations that are introduced. In a standard attack, any pixel can be modified. In an edges-only attack, the edges of objects in an image are perturbed. In a few-pixels attack, only a minimum amount of pixels can be manipulated. In general, the greater the restrictions on perturbations, the less the visual discrepancies seen in the modified images.

Figure 2 shows detection evasion attacks on NeuralHash using gradient-based modifications [25]. The top row of images shows the original image with targeted content and three modifications of the original image using standard, edges-only and few-pixels modification attacks, respectively. The row of text between the two rows of images shows the NeuralHash values of the original and modified images. The bottom row of images shows the differences between the original image and modified images where black corresponds to no pixel change and white corresponds to a pixel change.

Struppek et al. [25] define the optimization problem as:

$$\min_{X'} \; L_{\text{MSE}}(X', \widetilde{H}) - \lambda * \text{SSIM}(X', X) \quad \text{s.t. } d(H(X'), \widetilde{H}) > \delta_H$$

where $L_{\text{MSE}}()$ is a negative mean squared error (MSE) function used to increase the hash discrepancy between an image X and its manipulated counterpart X'

$(X, X' \in [-1, 1]^{h \times w \times c})$, $\widetilde{H} = H(X)$ and a threshold δ_H expresses the proportion of bits changed in the hash of the modified image.

To minimize visual perturbations, L_{MSE} is influenced by a penalty term based on the structural similarity index measure SSIM between the images weighted by a parameter λ. SSIM values range from zero to one, the closer the SSIM value is to one, the more similar the images. Readers are referred to [27] for details about SSIM. The optimization ends when the Hamming distance $d(H(X'), H(X))$ between the NeuralHash values of the original and perturbed images exceeds the threshold δ_H.

To evaluate the attacks, experiments were conducted on the first 10,000 samples of the ImageNet test split [23]. The optimization steps on the images were performed using Adam [12].

Struppek et al. [25] selected the success rate metric and the SSIM metric for visual similarity. However, they assessed the effort required to modify an image only in terms of the optimization steps required, not the time required for image modification.

All three levels of attacks created visually imperceptible evasion images. To explore the differences, Struppek et al. [25] proceeded to evaluated them using the metrics. The success rates were almost identical for the standard and edges-only attacks, 100% and 99.95%, respectively; in the case of the few-pixels attack, the success rate was slightly lower at 98.21%. The SSIM values were very similar. The standard, edges-only and few-pixels attacks yielded SSIM scores of 0.9999, 0.9996 and 0.9989, respectively. The optimization step metrics varied considerably – about five steps for the standard attack, about 150 steps for the edges-only attack and about 3,095 for the few-pixels attack. These results question whether the edges-only and few-pixels attacks provide any additional value compared with the standard attack. The results also demonstrate that it is very easy to achieve at least a small change in the NeuralHash value of an image.

Struppek et al. [25] only gave detailed metrics for $\delta_H = 0$ (i.e., images are perturbed until one bit in their hashes are flipped). For higher values of δ_H, they only published a single graph showing the relative changes in the evaluation results compared with those obtained with $\delta_H = 0$.

5.2 Hash Collision

A hash collision attack is similar to a second preimage attack in cryptographic hashing. The goal is to modify an image X showing non-targeted content in a way that alters its perceptual hash $H(X)$ to match the perceptual hash in the targeted content database [22, 25]. The perturbed image could be completely synthetic and not show any perceivable content or it could be based on a specific image that is manipulated while keeping its visual perception hash the same.

In this scenario, an attacker must be able to generate $H(X)$ and must know the perceptual hashes in the targeted content database. Access to the targeted content database may be controlled, but an attacker needs to know at least one hash that would lead to a collision.

A hash collision attack could be used in two ways. One use case is denial of service. The attack would generate and disseminate large numbers of non-targeted images that lead to hash collisions. This would induce a PHTCS system to produce numerous false positives, eventually overwhelming the system.

The second use case is framing innocent users. In this case, an attacker could produce collision images and send them to a victim. Depending on the policies in place for the PHTCS system, moderation actions against the victim could range from account deactivation to criminal prosecution. Groups of individuals could also be targeted by sending innocuous images whose hashes collide with those of images containing CSAM. All the members of the group who receive, store or disseminate the images would be flagged by the PHTCS system that is supposed to only flag CSAM [22, 25]. While collision attacks can be leveraged by a variety of actors, a key obstacle for non-government and low-resource actors is gaining access to the targeted content database.

PDQ Attacks Hash collision attacks can be developed to target Meta's PDQ. The attack starts with a known hash $H(X)$ of an image X in a targeted content database. The attack requires the hash $H(X)$, but if only X is known, $H(X)$ would probably be easy to compute. Additionally, an initial image S is required, which is transformed into a new image X' that is perceptually similar to S, but with $d(H(X), H(X')) < T$ where $d()$ computes the Hamming distance between the images and T is a preset threshold.

The perturbations that create image X' are identified by iteratively employing indirect Monte Carlo approximations of the hash function gradients that minimize loss terms over hash distances and perturbation sizes [22]. The perturbations should be minimal so that they do not interfere with the visual perception of the image [22]. Each iteration would identify a perturbation vector δ that makes minimal changes to X' while decreasing the Hamming distance $d(H(X), H(X' + \delta))$. This vector would then be added to X' and the iterative process repeated until $d(H(X), H(X')) < T$ holds.

Since PDQ is not differentiable, the gradient must be estimated for each iteration to proceed. At each step, $H(X'_i)$ is computed for the current candidate image X'_i as well as Δ_i, the current distance to the target hash. Next, q perturbations $p_1, ..., p_q$ are generated where q determines the accuracy of the approximation. In [22], the size changes based on the iteration number. For each perturbation, the change in the hash value c_j is computed according to the target function [22]:

$$c_j = d(H(X), H(X'_i + p_j)) - d(H(X), H(X'_i))$$

Next, the changes c_j are used to compute a weighted average over the perturbations [22]:

$$\delta'_i = \frac{1}{q} \sum_{1}^{q} c_j * p_j$$

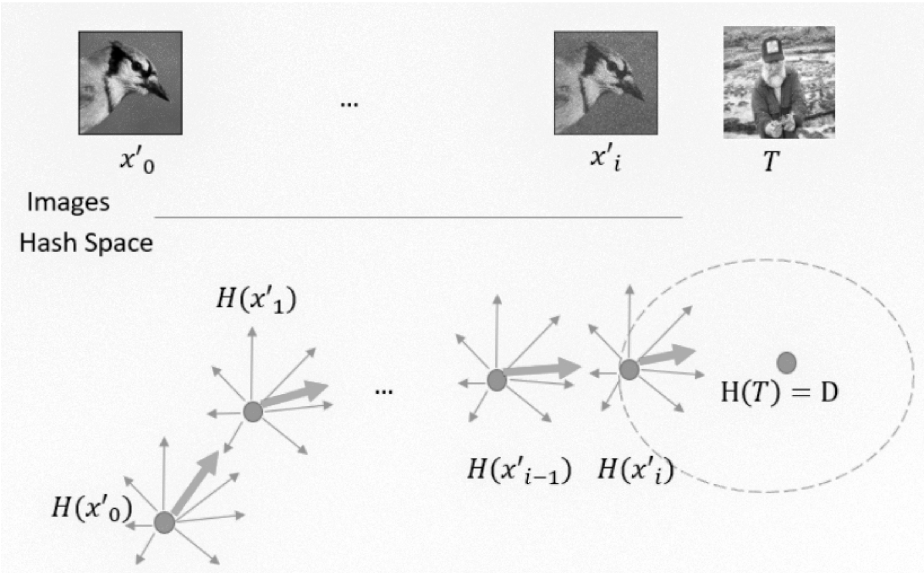

Fig. 3. Gradient estimations leading to a PDQ hash collision (from [22]).

Finally, the gradient δ_i used in the iteration is obtained by normalizing δ'_i and multiplying it by the learning rate λ [22].

Figure 3 shows the iterative process of estimating gradients that lead to collisions. The top portion of the figure shows the changes to the images up to the colliding target image. Below the images are the steps in hash space. The thick arrows in the figure denote the gradients used to move closer to the colliding hash during successive iterations [22].

Prokos et al. [22] used a threshold T of 90 in their experiments. However, in a real system the threshold setting would depend on various factors, especially the likelihood of false positives. The experiments were performed with 30 image pairs randomly selected from the ImageNet validation set [23]. All 30 image pairs had PDQ hash collisions within the threshold, requiring up to 6,350 iterations. The execution time for finding collision hashes for all 30 pairs was about three hours [22].

Figure 4 shows a hash collision attack on PDQ over several iterations [22]. The progression of images in the top row from left to right starts with an image of a bird. Successive images get more perturbed while the Hamming distance between the PDQ hashes of each successive image and the original image is reduced, until the final image of a man is obtained. The bottom row of images shows the differences between the original image and modified images where black corresponds to no pixel change and white corresponds to a pixel change.

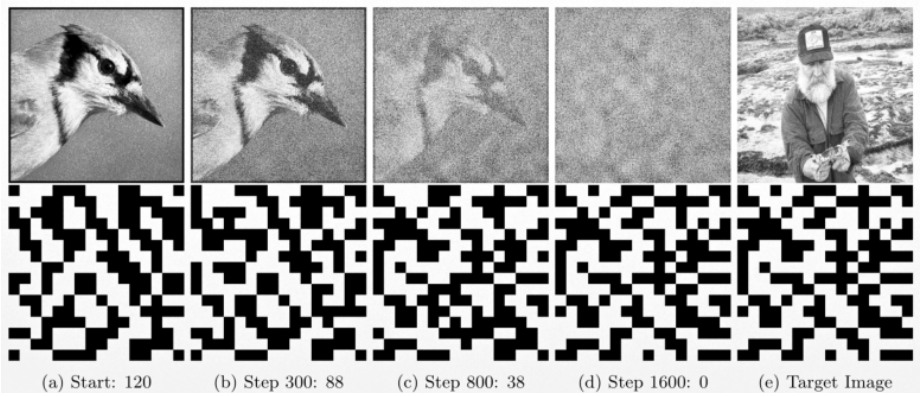

Fig. 4. Hash collision attack on PDQ (from [22]).

5.3 Data Leakage

A data leakage attack seeks to extract information from a perceptual hash. A perceptual hash function extracts image features and converts them to a hash value. This means that a perceptual hash contains some image information [25]. In order to extract information, an attacker would have to collect as many perceptual hashes as possible (to correlate the findings) and would need access to and details about the perceptual hash function [25].

The severity of data leakage depends on how much information can be extracted from a hash. A data leakage attack is less severe if the extracted data only indicates attributes of the original image. Thus, an attacker could derive some general statements about the original image, such the image shows a dog or contains two humans. In a more severe attack, it would be possible to reconstruct portions of the original image from its hash, such as a thumbnail version [22]. Of course, since perceptual hashing compresses a variable-sized image to a fixed size, only a low resolution version of the original image could be reconstructed.

In addition to the amount of information that can be extracted, it is important to consider the potential victims of data leakage. A PHTCS system processes perceptual hashes of two types of images, user images and targeted content images. Both types of perceptual hashes should be protected. Safeguarding user hashes protects user privacy even if only limited image information can be extracted [25]. Therefore, user hashes should always reside on user devices and not be transmitted to servers unless there are targeted content matches. When user hashes are transmitted to servers, the hashes should be safeguarded on the server-side so that only authorized access is possible.

Information leakage from targeted content hashes that enables images to be reconstructed would have disastrous consequences [22]. Therefore, targeted content hashes should have strong protections. Moreover, even during client-side scanning, the hashes should never be stored on user devices. Apple [2, 6] and Kulshrestha and Mayer [16] have provided suggestions about how such a system might work.

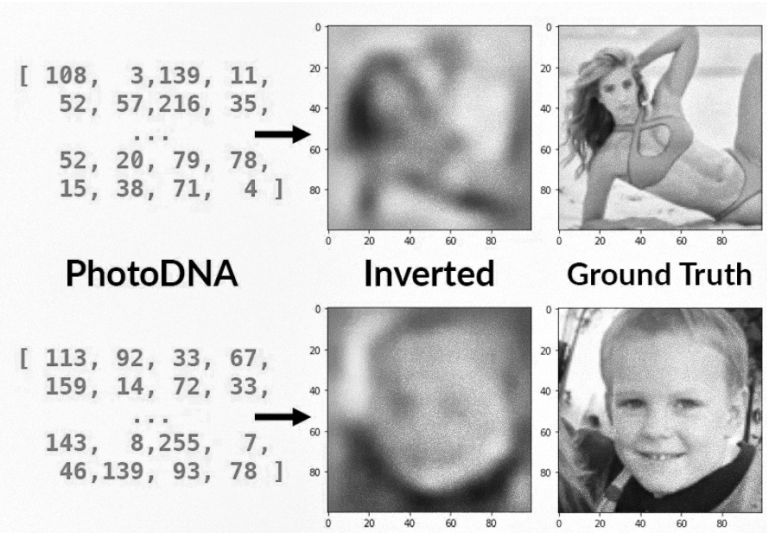

Fig. 5. Image reconstructions from PhotoDNA hashes [4].

PhotoDNA Attack Research on reconstructing images from perceptual hashes is sparse. However, an informal research article in the form of a blog post [4] discusses the reconstruction of images from Microsoft's PhotoDNA perceptual hashes [19]. Since PhotoDNA is not published and can only be used under a strict non-disclosure agreement, the data leakage attack was developed by reverse engineering the PhotoDNA algorithm.

Specifically, the data leakage attack employs a trained neural network that takes a PhotoDNA hash as input and generates a representation of the original image [4]. The neural network training dataset comprised images and their PhotoDNA hashes. In fact, the data leakage attack only became possible because the compiled PhotoDNA code became available and was reverse-engineered to compute PhotoDNA hashes of training images for the training dataset [4]. The results of image reconstruction strongly depend on the similarity between the training dataset and the unknown images to be reconstructed. For example, reconstructing images with faces of celebrities would work best if the neural network was trained on the same type of images [4].

Figure 5 shows examples of image reconstruction (inverted images) from PhotoDNA hashes [4]. The reconstructions, which are very blurry, are deemed "good results." In many cases, the results are much worse [4]. Thus, it is questionable if a data leakage attack could identify a user from a user hash or produce a detailed image from a targeted content hash.

5.4 Evaluation of NeuralHash Detection Evasion

Unfortunately, Struppek et al. [25], who devised the detection evasion attacks on NeuralHash, did not provide detailed evaluations of their results. Specifically,

Table 1. Expanded evaluation of detection evasion attacks on NeuralHash.

δ_H	0	0.05	0.1	0.15
SR	100%	100	100%	100%
Success	10,000	10,000	10,000	10,000
Failure	0	0	0	0
SSIM (Av. & SD)	0.9998 ± 0.000	0.9991 ± 0.002	0.9976 ± 0.004	0.9953 ± 0.007
Maximum	0.9999	0.9999	0.9998	0.9997
Minimum	0.9894	0.9593	0.9097	0.8675
Time (s) (Av. & SD)	3.5 ± 3.3	15.1 ± 14.4	26.7 ± 19.7	38.7 ± 24.2
Maximum	71.1	152.1	203.3	308.3
Minimum	1.5	1.7	1.8	2.5
Steps (Av. & SD)	5.4 ± 4.9	20.65 ± 19.5	42.3 ± 29.2	61.9 ± 35.6
Maximum	102	294	338	393
Minimum	4	4	4	4
Evaluation Time (h)	3.9	13.4	24	34.6

they computed the success rate (SR) and SSIM metrics, but did not provide information about the time requirements for optimizing their gradient-based detection evasion attacks on NeuralHash. The average optimization steps required were provided, but no information about the time required for single steps. The evaluation also focused on finding minimal hash changes (i.e., one flipped bit). Additionally, a brief overview was provided about how the SR and SSIM values and number of optimization steps would change with larger differences in the hash values.

This section builds on the work of Struppek et al. [24] by verifying their SR and SSIM results, expanding their evaluation to include the time needed to create detection evasion images and evaluating the attack efficacy for larger minimum Hamming distances δ_H between the NeuralHash values of the original and modified images. The code provided by Struppek et al. was modified slightly to include time measurements in the logging. Next, the logged metrics were computed and reported. The same image dataset (i.e., first 10,000 images from ImageNet ILSVRC2012 test split [23]) and experimental setup were employed.

Table 1 shows the expanded evaluation of the detection evasion attacks on NeuralHash (see Table 2 in [25] for a direct comparison). The experiments used different Hamming distance thresholds δ_H from the set $\{0, 0.05, 0.1, 0.15, 0.2\}$. Note that threshold $\delta_H = 0$ is the default attack mode where an image is perturbed until only one bit in the hash of the modified image is changed. The threshold $\delta_H = 0.1$ means that at least 10% of the hash bits are flipped. The SR

results are provided as percentages as well as the absolute numbers of successful and unsuccessful attacks in the dataset. For the other three metrics, SSIM, time and steps, the average, standard deviation, and maximum and minimum values are provided.

The results show that across the range of experiments, all 10,000 images were successfully perturbed to evade detection determined by varying the δ_H threshold. Furthermore, the average SSIM values are consistently very high over all the experiments. Even in the experiment with $\delta_H = 0.15$, the average SSIM value obtained is 0.9953 with a small deviation, which would correspond to a nearly indistinguishable perturbed image. However, some outliers do exist, starting in the experiments with $\delta_H = 0.1$. These outliers would probably show visible perturbations because the minimum SSIM value drops to 0.9097 and even down to 0.8675 for $\delta_H = 0.15$. The SR and SSIM results in Table 1 confirm the results presented by Struppek et al. [25].

Table 1 shows that the time required for image perturbations increases considerably for high δ_H values. The increase in the time requirement as δ_H goes from zero to 0.05 is large and continues to rise at an average of 38.7 s per image when $\delta_H = 0.15$. The time requirements deviate greatly from the average, which means that the images are very different in how easily they can be perturbed. The trend is also seen in the minimum and maximum values that strongly deviate from the average values. The attack on a single image, in the case of $\delta_H = 0.15$, requires a maximum of 308 s (more than five minutes), which is a long time.

Unfortunately, no patterns are discerned that would provide insights into what makes images easy or hard to perturb in the experimental setup. However, an average of 38.7 s to perturb an image is not very high considering the attack scenarios. The last row in Table 1 lists the evaluation times to provide a sense of how much time an attacker would need to perturb a large image dataset as used in the evaluation. Indeed, the evaluation times show that a determined attacker is definitely capable of executing detection evasion attacks on large datasets. Of course, the time measurements are strongly influenced by the computational resources used in the experiments. Fortunately, the downloadable code available at [25] can execute on graphical processor units (GPUs), which would speed up successful attack development.

6 Conclusions

The dissemination of CSAM on the Internet is a serious problem and technological approaches are required to combat its spread and enable the prosecution of perpetrators. However, it is important to maintain user security and privacy while enabling law enforcement to scan images for illegal content. PHTCS has been suggested as a solution that balances these two requirements. This chapter has discussed the state-of-the-art in PHTCS and conducted an extensive evaluation of potential attacks. Certain problems must be addressed to enable effective PHTCS deployments that would be acceptable to all the stakeholders.

A key problem raised by this research is that PHTCS effectiveness is severely limited by attacks such as detection evasion, hash collision and data leakage. A skilled individual could create and disseminate tools that manipulate CSAM to evade detection, rendering PHTCS systems essentially useless. Hash collision attacks could be used to create innocuous images with the same perceptual hashes as known CSAM images. Large numbers of these innocuous images could be disseminated to cause PHTCS to raise alerts, eventually overwhelming the system to result in denial of service. More insidious is the possibility that innocuous images could be used to frame individuals and groups with possessing and disseminating CSAM, exposing them to criminal investigations and potential prosecution. Data leakage attacks, which invert perceptual hashes to obtain portions or low-resolution versions of the original images, pose privacy risks to subjects who appear in the original images as well as to users. Clearly, additional research is needed to harden perceptual hashes and render them attack-resistant.

Another problem is that perceptual hashing technologies and processes and policies for their use are often developed and deployed in secret. Tech companies, non-governmental organizations, law enforcement, other government agencies and even researchers are cautious about disclosing details fearing that they may help perpetrators evade exposure. While this is understandable, it hinders informed discussion and the lack of transparency spreads mistrust in the research community and user base. A PHTCS solution that balances child safety and security and user privacy concerns can only become operational with mutual trust and collaboration. It is hoped that this chapter will stimulate discussion, research and, eventually, viable PHTCS deployments.

References

1. H. Abelson, R. Anderson, S. Bellovin, J. Benaloh, M. Blaze, J. Callas, W. Diffie, S. Landau, P. Neumann, R. Rivest, J. Schiller, B. Schneier, V. Teague and C. Troncoso, Bugs in our pockets: The risks of client-side scanning, arXiv: 2110.07450v1 (`arxiv.org/abs/2110.07450v1`), 2021.
2. Apple, CSAM Detection – Technical Summary, Cupertino, California (`www.apple.com/child-safety/pdf/CSAM_Detection_Technical_Summary.pdf`), 2021.
3. Apple, Security Threat Model Review of Apple's Child Safety Features, Cupertino, California (`www.apple.com/child-safety/pdf/Security_Threat_Model_Review_of_Apple_Child_Safety_Features.pdf`), 2021.
4. A. Athalye, Inverting PhotoDNA, *Internet Archive*, December 20, 2021.
5. A. Athalye, NeuralHash Collider, GitHub (`github.com/anishathalye/neural-hash-collider`), 2023.
6. A. Bhowmick, D. Boneh, S. Myers, K. Talwar and K. Tarbe, The Apple PSI System, Apple, Cupertino, California (`www.apple.com/child-safety/pdf/Apple_PSI_System_Security_Protocol_and_Analysis.pdf`), 2021.
7. F. Breitinger, B. Guttman, M. McCarrin, V. Roussev and D. White, Approximate Matching: Definition and Terminology, NIST Special Publication 800-168, National Institute of Standards and Technology, Gaithersburg, Maryland, 2014.
8. Facebook, Online child protection – Tools and technology, *Internet Archive*, June 21, 2022.

9. Federal Criminal Police Office, Presentation of the Numbers of Child Victims of Violence – Evaluation of the Police Crime Statistics 2021 (in German), Wiesbaden, Germany (`www.bka.de/SharededDocs/Downloads/DE/AktuelleInformationen/Infografiken/Sonstige/kindlicheGewaltopfer_PKS2021.pdf`), 2022.
10. Q. Hao, L. Luo, S. Jan and G. Wang, It's not what it looks like: Manipulating perceptual-hashing-based applications, *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, pp. 69–85, 2021.
11. S. Jain, A. Cretu and Y. Montjoye, Adversarial detection avoidance attacks: Evaluating the robustness of perceptual-hashing-based client-side scanning, *Proceedings of the Thirty-First USENIX Security Symposium*, pp. 2317–2334, 2022.
12. D. Kingma and J. Ba, Adam: A method for stochastic optimization, poster paper presented at the *Third International Conference on Learning Representations*, 2015.
13. E. Klinger and D. Starkweather, pHash: The open source perceptual hash library, *Internet Archive*, April 9, 2023.
14. N. Krawetz, Looks like it, *Internet Archive*, May 26, 2011.
15. N. Krawetz, Kind of like that, *Internet Archive*, January 21, 2013.
16. A. Kulshrestha and J. Mayer, Identifying harmful media in (end-to-end) encrypted communications: Efficient private membership computation, *Proceedings of the Thirtieth USENIX Security Symposium*, pp. 893–910, 2021.
17. Meta, Open-sourcing photo- and video-matching technology to make the Internet safer, *Internet Archive*, August 1, 2019.
18. Meta, The TMK+PDQF Video-Hashing Algorithm and the PDQ Image Hashing Algorithm, Menlo Park, California (`raw.githubusercontent.com/facebook/ThreatExchange/main/hashing/hashing.pdf`), 2021.
19. Microsoft, Photo DNA, *Internet Archive*, April 11, 2023.
20. National Center for Missing and Exploited Children, CyberTipline 2021 Report, *Internet Archive*, March 17, 2022.
21. D. Petrov, Wavelet image hash in Python, *Internet Archive*, July 2, 2016.
22. J. Prokos, T. Jois, N. Fendley, R. Schuster, M. Green, E. Tromer and Y. Cao, Squint hard enough: Evaluating perceptual hashing with machine learning, *Cryptology ePrint Archive*, paper no. 2021/1531 (`eprint.iacr.org/2021/1531`), 2021.
23. O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. Berg and F. Li, ImageNet large scale visual recognition challenge, *International Journal of Computer Vision*, vol. 115(3), pp. 211–252, 2015.
24. L. Struppek, Learning-to-Break-Deep-Perceptual-Hashing, GitHub (`github.com/ml-research/Learning-to-Break-Deep-Perceptual-Hashing`), 2022.
25. L. Struppek, D. Hintersdorf, D. Neider and K. Kersting, Learning to break deep perceptual hashing: The use case NeuralHash, *Proceedings of the ACM Conference on Fairness, Accountability and Transparency*, pp. 58–69, 2022.
26. K. Walker, Four steps we're taking today to fight terrorism online, *Internet Archive*, June 18, 2017.
27. Z. Wang, A. Bovik, H. Sheikh and E. Simoncelli, Image quality assessment: From error visibility to structural similarity, *IEEE Transactions on Image Processing*, vol. 13(4), pp. 600–612, 2004.
28. WeProtect Global Alliance, Global Threat Assessment 2021, Sevenoaks, United Kingdom (`www.weprotect.org/wp-content/uploads/Global-Threat-Assessment-2021.pdf`), 2021.

29. B. Yang, F. Gu and X. Niu, Block mean value based image perceptual hashing, *Proceedings of the International Conference on Intelligent Information Hiding and Multimedia*, pp. 167–172, 2006.
30. C. Zauner, Implementation and Benchmarking of Perceptual Image Hash Functions, M.S. Thesis, Secure Information Systems Program, University of Applied Sciences Upper Austria, Hagenberg, Austria, 2010.