Automated Side Channel Analysis of Media Software with Manifold Learning

Yuanyuan Yuan, Qi Pang, Shuai Wang*

The Hong Kong University of Science and Technology
{yyuanaq, qpangaa, shuaiw}@cse.ust.hk

1 Introduction

Side channel analysis (SCA) infers program secrets by analyzing the target software's influence on physical computational characteristics, such as the execution time, accessed cache units, and power consumption. With the adoption of cloud computing and machine learning as a service (MLaaS), media software, a type of application software used for processing media files like images and text, is commonly involved in processing private data uploaded to cloud (e.g., for medical diagnosis). Existing works have exploited media software with extensive manual efforts or reconstruct only certain media data [12, 44, 47]. However, the community lacks a systematic and thorough understanding of SCA attack vectors for media software and of the ways that private user inputs of various types (e.g., images or text) can be reconstructed in a unified and automated manner. Hence, this is the first study toward media software of various input formats to assess how their inputs, which represent private user data, can be leaked via SCA in a fully automatic way.

Recent advances in representation learning and perceptual learning [3, 49] inspired us to recast SCA of media software as a cross-modality manifold learning task in which an autoencoder [17] is used to learn the mapping between confidential media inputs and the derived side channel traces in an end-to-end manner. The autoencoder framework can learn a low-dimensional joint manifold of media data and side channel observations to capture a highly expressive representation that is generally immune to noise.

Our proposed autoencoder framework is highly flexible. It converts side channel traces into latent representations with an encoder module ϕ_θ, and the media data in image, audio and text formats can be reconstructed by assembling decoders ψ_θ that correspond to various media data formats to ϕ_θ. By enhancing encoder ϕ_θ with attention [43], the autoencoder framework can automatically localize program points that primarily contribute to the reconstruction of media inputs. That is, the attention mechanism delivers a "bug detector" to locate program points at which information can leak.

The observation that manifold learning captures key perceptions of high-dimensional data in a low-dimensional space [49] inspired us to propose the use of *perception blinding* to mitigate manifold learning–based SCA. Well-designed perception blinding "dominates" the projected low-dimensional perceptions and thus confines adversaries to only generate media data perceptually bounded to the mask. In contrast, media software that is typically used to process data bytes of media data experiences no extra difficulty in processing the blinded data and recovering the original outputs.

Tool and Data. We released our tool and all data at https://github.com/Yuanyuan-Yuan/Manifold-SCA [2]. Our tool passed the artifact evaluation of USENIX Security 2022 and got all badges (i.e., available, functional, and reproduced).

Extended Version. We provided an extended version of this paper at https://arxiv.org/pdf/2112.04947.pdf [1] which includes full details of our study.

2 Threat Model and Attack Scenarios

This study reconstructs confidential inputs of media software from side channels. We thus reasonably assume that different inputs of targeted media software can induce distinguishable memory access traces. Otherwise, no information regarding inputs would be leaked.

Profiled SCA [14–16, 20, 29] commonly assumes that side channel logs have been prepared for training and data reconstruction. For our scenario, we generally assume a standard *trace-based* attacker. We assume that a trace of system side channel accesses made by the victim software has been prepared for use. Our evaluated system side channels include cache line, cache bank, and page table entries. The feasibility of logging such fine-grained information has been demonstrated in real-world scenarios [8, 12, 44, 46], and this assumption has been consistently made by many previous works [4, 10, 19, 38, 39, 42]. In this study, we use Intel Pin [28]

*Corresponding author.

to log memory access traces and convert them into corresponding side channel traces (see details in [1]).

We also benchmark userspace-only scenarios where attackers can launch `Prime+Probe` attack [36] to log cache activities when media software is processing a secret input. We use Mastik [45], a micro-architectural side channel toolkit, to launch "out-of-the-box" `Prime+Probe` and log victim's L1I and L1D cache activities. We pin victim process and `spy` process on the same CPU core; see attack details in [1].

Exploiting new side channels is *not* our focus. We demonstrate our attack over commonly-used side channels. This way, our attack is shown as practically approachable, indicating its high impact and threats under real-world scenarios. Unlike previous SCA on media software [12, 44] or on crypto libraries [48], we do *not* require a "white-box" view (i.e., source code) of victim software. We automatically analyze media software *executables* with different input types. We present some examples in Sec. 5 and more results are given in [2] and [1]. In brief, the reconstructed media data manifest excellent (visual) similarity to user inputs. Many studies have only flagged program points of information leakage with (unscalable) abstract interpretation or symbolic execution [4, 9, 39]. Direct reconstruction of media data is beyond the scope of such formal method-based techniques, and these studies did not propose SCA mitigation.

3 A Manifold View on SCA of Media Software

This study recasts the SCA of media software as a cross-modality manifold learning task that can be well addressed with supervised learning. We train an autoencoder [17] that maps side channel observations O to the media inputs I of media software. Our threat model (Sec. 2) assumes that attackers can profile the target media software and collect side channel traces derived from many inputs. Therefore, our autoencoder framework is trained to learn from historical data and implicitly forms a low-dimensional joint manifold between the side channel logs and media inputs. We first introduce the concept of manifold, which will help to clarify critical design decisions of our framework (see Sec. 4).

Manifold Learning. The use of manifold underlies the feasibility of dimensionality reduction [22]. The key premise of manifold is the *manifold hypothesis*, which states that real-world data in high-dimensional space are concentrated near a low-dimensional manifold [11]. That is, real-world data often lie in a manifold $\mathcal{M}$ of much lower dimensionality d, which is embedded in its high-dimensional space $\mathcal{R}$ of dimensionality D ($d \ll D$). *Manifold learning* aims to find a projection $f : \mathcal{R} \to \mathcal{M}$ that converts data $x \in \mathcal{R}$ into y in an intrinsic coordinate system of $\mathcal{M}$.[1] f^{-1} projects $f(x) \in \mathcal{M}$ back onto representation x in the high-dimensional space $\mathcal{R}$.

[1] "Intrinsic coordinate" denotes the coordinate system of the low-dimensional manifold space for each high-dimensional data sample [7, 25].

Manifold learning views high-dimensional media data $x \in \mathcal{R}$ as a composite of perceptually meaningful contents that are shown as robust to noise or other input perturbations [3, 49, 50]. Manifold learning algorithms extract expressive representations of high-dimensional data such as images, audio, and text [13], which explains why AI models can make accurate predictions pertaining to high-dimensional data [3].

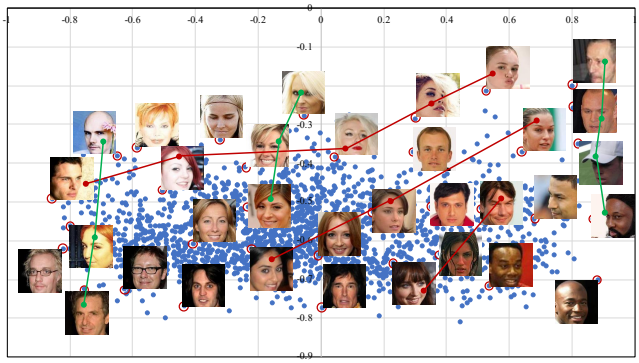

Figure 1: Project face photos to a 2-dimensional manifold.

In Fig. 1, we project a set of real-world face images to manifold $\mathcal{M}_{img}$ of two dimensions. To draw this projection, we tweak our autoencoder framework (see Sec. 4) to convert each face image into a latent representation of two dimensions. We observe that images are generally separated by skin and hair colors; in addition, face orientations are roughly decomposed into two orthogonal directions (green and red lines). In fact, recent studies have shown the effectiveness of conducting image editing by first projecting image samples into the manifold space [50]. Editing images in the high-dimensional space involves a large search space $[0, 255]$ for every pixel; the random selection of pixel values from $[0, 255]$ struggles to create realistic images because arbitrary editing could "fall off" the manifold of natural images. In contrast, manifold learning facilitates sampling within $\mathcal{M}$, and the perceptually meaningful contents in $\mathcal{M}$ confine manipulations to generate mostly realistic images [50].

Parametric Manifold. Most manifold learning schemes adopt non-parametric approaches. Despite the simplicity, non-parametric approaches cannot be used to project *new* data points in $\mathcal{R}$ onto $\mathcal{M}$. Recent advances in deep neural networks, particularly autoencoders, have enabled a parametric nonlinear manifold projection $f_\theta : \mathcal{R} \to \mathcal{M}$ [49]. Manifold learning can thus process unknown data points of high-dimensional media data.

High-Level Research Overview

Processing media data has an observable influence on the underlying computing environment; it thus induces side channel traces that can be logged by attackers to infer private inputs. Previous SCA studies, from a holistic view, attempted to (manually) map side channel logs to *data bytes* in media data (similar to how media software treats media data) [12, 44];

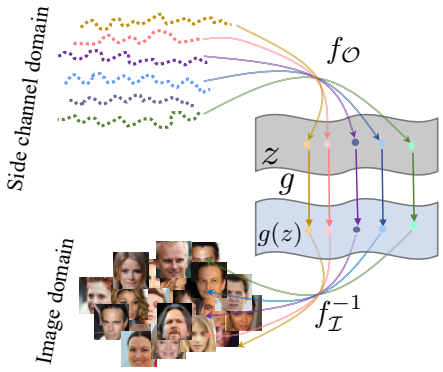

Figure 2: Mapping between side channels and images via a low-dimensional joint manifold $\mathcal{M}_{\mathcal{I},\mathcal{O}} = \mathcal{I} \times \mathcal{O}$.

reconstruction of media data in a per-pixel manner is thus error-prone and likely requires expertise and manual efforts.

The success of manifold learning in tasks like image editing and cross-modality reconstruction [49, 50] led us to construct a joint, *perception-level* connection between side channel logs and high-dimensional media inputs.[2] Therefore, instead of deciding value of each byte, reconstructing media data is recast into exploring the manifold of media data that satisfies the perception-level combinatory constraints.

Overall, we view SCA as a cross-modality high-dimensional data reconstruction task that is addressed with joint manifold learning in this work [49]. Let the media software be P and its inputs be I; the attacker's observation on executing each input $i \in I$ can be represented as $(view \circ P) : I \to O$, where $o \in O$ denotes the observation of side channel traces. Let F be the composite function $view \circ P$. According to the manifold hypothesis, we assume that I and O also lie in the unknown manifolds $\mathcal{I}$ and $\mathcal{O}$, respectively. As mentioned in our threat model (Sec. 2), we assume that side channel observations depend on the inputs of media software; therefore, the entire joint dataset $\{i_i, o_i\}$ formed by the ith media input $i_i \in I$ and the corresponding ith observation $o_i \in O$ lies in a joint manifold

$$\mathcal{M}_{\mathcal{I},\mathcal{O}} = \{(i, F(i)) | i \in \mathcal{I}, F(i) \in \mathcal{O}\}$$

where $(i, F(i))$ is described with the regular high-dimensional coordinate system. Since I and O also lie in the corresponding manifolds $\mathcal{I}$ and $\mathcal{O}$, the data points in $\mathcal{M}_{\mathcal{I},\mathcal{O}}$ should be equivalently described using an intrinsic coordinate system $(z, g(z))$. Hence, we assume the existence of a homomorphic mapping $(f_{\mathcal{I}}, f_{\mathcal{O}})$ over $(z, g(z))$ such that $z = f_{\mathcal{O}}(o)$ and $g(z) = f_{\mathcal{I}} \circ F^{-1}(o)$. $f_{\mathcal{O}}$ maps side channel observation $\mathcal{O}$ onto the intrinsic coordinate z, whereas $f_{\mathcal{I}}$ maps high-dimensional media data I onto $g(z)$. Note that g denotes the diffeomorphism (i.e., an isomorphism of two manifolds) between the $\mathcal{I}$ and $\mathcal{O}$ manifolds [49]. Hence, instead of computing $F^{-1} = (view \circ P)^{-1} : O \to I$ to map the side channel observation back onto the media inputs, we leverage the joint manifold to constitute the following composite function:

$$F^{-1}(o) = f_{\mathcal{I}}^{-1} \circ g \circ f_{\mathcal{O}}(o) \quad (1)$$

$i \in I$ can thus be reconstructed using the inverse composite function $f_{\mathcal{I}}^{-1} \circ g \circ f_{\mathcal{O}}$ over the joint manifold $\mathcal{M}_{\mathcal{I},\mathcal{O}}$. Fig. 2 provides a summary and presents a schematic view of how I and O of high-dimensional data are mapped via $\mathcal{M}_{\mathcal{I},\mathcal{O}}$.

The feasibility of using neural networks, especially autoencoders, to facilitate parametric manifold learning has been discussed [18, 30, 49, 50]. Accordingly, we train an autoencoder by encoding side channel traces O onto the latent space with encoder ϕ_θ and by generating media data I with decoder ψ_θ from the latent space. Therefore, Eq. 1 can be learned in an end-to-end manner [3, 49]. Holistically, ϕ_θ and ψ_θ correspond to $f_{\mathcal{O}}$ and $f_{\mathcal{I}}^{-1}$, respectively, whereas g is implicitly constructed in the encoded latent space.

4 Framework Design

We describe the design of our autoencoder in Sec. 4.1. Sec. 4.2 clarifies the usage of attention to localize code fragments inducing information leakage. Sec. 4.3 introduces perception blinding to mitigate our SCA.

4.1 SCA with Autoencoder

We propose a general and highly-flexible design in which an autoencoder is used to facilitate SCA of various media data, including images, audio and text. The autoencoder framework [17] defines a parametric feature-extracting function f_θ, named *encoder*, that enables the projection of the input x onto a latent vector $h = \phi_\theta(x)$. Similarly, autoencoder frameworks also use ψ_θ as a *decoder* that reconstructs input $\hat{x}$ from a latent vector $\hat{x} = \psi_\theta(h)$. A well-trained autoencoder framework gradually identifies a parameter vector θ to minimize the reconstruction error as follows:

$$L(\theta) = \sum_t L(x_t, \psi_\theta \circ \phi_\theta(x_t))$$

where x_t is a training sample. Minimal errors can be found with statistical methods like stochastic gradient descent.

The first row of Fig. 3 depicts the workflow. We clarify that our focus is *not* to propose novel model architectures; rather, we show that high-quality inputs can be synthesized by assembling standard models, which indicates severity and effectiveness of our attack. We now discuss the high-level workflow (see model structures and training details in [1]). Given a logged side channel trace $o \in O$, encoder $\phi_\theta(o)$ converts o into the corresponding latent representation. We prepare three decoders ψ_θ^i, ψ_θ^a, ψ_θ^t to reconstruct these types of media data (i.e., image, audio, and text) from the encoded latent representation. We pair encoder $\phi_\theta(o)$ with each ψ_θ^* and train the assembled pipeline for our customized objective functions $L(\theta)$. Our proposed framework is task-agnostic. Generating

[2] Perception-level connection means constraints on data bytes formed by perceptual contents (e.g., gender, hair style) in media data are extracted from side channels.

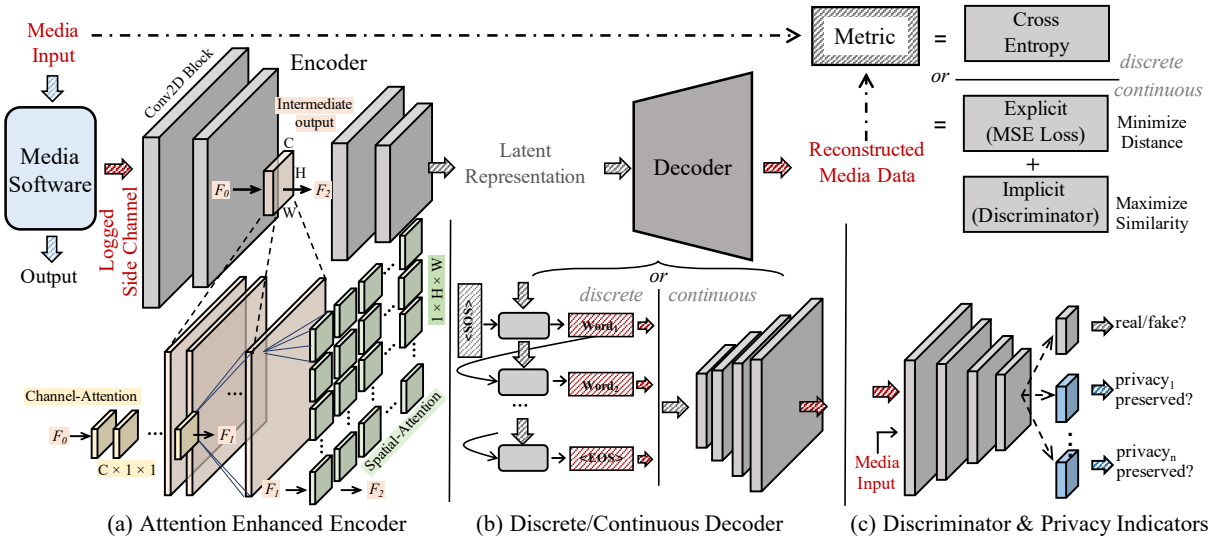

Figure 3: Reconstructing media data of different types with a unified autoencoder framework.

media data of various types requires only assembling corresponding decoders to the unified encoder ϕ_θ.

Encoder ϕ_θ. A logged side channel trace will first be folded into a $K \times N \times N$ matrix (see Table 2 for the detailed configuration of each trace). We then feed this matrix as the input of encoder ϕ_θ. The encoder ϕ_θ comprises several stacked 2D convolutional neural networks (CNNs). For the current implementation, ϕ_θ converts the high-dimensional inputs into latent vectors of 128 dimensions, given that the dimensions of our media inputs are all over 10K. We visualize encoding of side channel traces, including traces collected from both Intel Pin and `Prime+Probe`, in the extended version of this paper [2]. Moreover, we find that increasing the dimension of latent vectors (i.e., from 128 to 256) does not make an observable improvement. This observation is consistent with the manifold hypothesis [22], such that only *limited* "perceptions" exist in normal media data. In contrast, reducing the number of dimensions (e.g., 32) makes the outputs (visually) much worse. However, users who strive to recover media data of lower-dimensions can configure our framework with smaller latent vectors (e.g., 32 dimensions).

Fig. 2(a) shows that we enhance encoder ϕ_θ with attention. Indeed, we insert one attention module between every two stacked CNN layers in the encoder. Attention generally improves output quality of autoencoder [37]. More importantly, attention facilitates localizing program points of information leakage. We elaborate on Fig. 2(a) in Sec. 4.2.

Decoder ψ^*_θ. We categorize the media data exploited by this study into two types: continuous and discrete. Image and audio data are represented as a continuous floating-point matrix and reconstructed by ψ^i_θ and ψ^a_θ in a continuous manner. In contrast, textual data comprise word sequences, and because there is no "intermediate word," textual data are regarded as sequences of discrete values and handled by ψ^t_θ.

As shown in Fig. 2(b), we use a common approach to stacking 2D CNNs to design ψ^i_θ. A 2D CNN has several convolutional kernels; each kernel focuses on one feature dimension of its input and captures the spatial information of this feature dimension. Images can thus be reconstructed from vectors in the low-dimensional latent space with stacked 2D CNNs, as each 2D CNN upsamples from the output of the previous layer. For audio data, we first convert raw audio into the log-amplitude of Mel spectrum (LMS), a common 2D representation of audio data. This way, audio data are represented as 2D images (see some examples in [2]), in which the x-axis denotes time and the y-axis denotes the log scale of amplitudes at different frequencies. Herein, like ψ^i_θ, ψ^a_θ uses stacked 2D CNNs to process each converted 2D image, gradually upsamples from the latent representation, and reconstruct the LMSs of the audio data. Because the LMSs usually are not in square-shape, we append a fully connected layer to transform the shape of the reconstructed LMSs. These LMSs are then converted to raw audio losslessly.

Textual data, however, are reconstructed sequentially "word by word" due to their discrete nature. As shown in Fig. 2(b), to reconstruct a sentence from the latent space of a side channel trace o, a single word is gradually inferred based on words already inferred from sentence i. Following a common practice of training sequence-to-sequence autoencoders, we add a start-of-sequence (SOS) token before each sentence i and an end-of-sequence (EOS) token after i. Then, given a side channel trace o that corresponds to unknown text i, the trained decoder ψ^t_θ starts from the SOS token and predicts a word $w \in i$ sequentially until it yields the EOS token. From a holistic perspective, the trained model projects a sentence i into a low-dimensional manifold space of *word dependency*, which facilitates the gradual inference of each word w on i.

Designing Objective Functions. As depicted in the first row of Fig. 3, for discrete data (i.e., text), each decode step is a

Table 1: Privacy-aware indicators. Table 2 introduces each dataset.

Dataset	Indicator
CelebA	Is the celebrity's identity preserved?
ChestX-ray	Is the disease information preserved?
SC09	Is the speaker's identity preserved?
Sub-URMP	Is the musical instrument's type preserved?

multi-class classification task where the output is classified as one element in a pre-defined dictionary. Thus, we use *cross entropy* as the training objective. For continuous data, we design the training objective L_θ *composing both explicit and implicit metrics*. We now introduce each component in detail.

Explicit Metrics A common practice in training an autoencoder is to explicitly assess the point-wise distance between the reconstructed media input i' and reference input i with metrics such as MSE loss, L1 loss, and KL divergence [6, 21]. The autoencoder will be guided to gradually minimize the point-wise distance $L_\theta(i, \psi_\theta \circ \phi_\theta(o))$ during training. Nevertheless, a major drawback of such explicit metrics is that the loss of each data point is calculated *independently* and contributes *equally* to update θ and minimize L_θ. Our preliminary study [2] shows that such explicit metrics suffer from "over-smoothing" [33], a well-known problem that leads to quality degradation of the reconstructed data.

Implicit Metrics Another popular approach is to assess the "distributed similarity" [33] of reconstructed i' and reference input i. Viewing the general difficulty of extracting the distribution of arbitrary media data, a common practice is to leverage a neural discriminator D. Discriminator D and decoder ψ_θ play a zero-sum game, in which D aims to distinguish the reconstructed input i' from normal media data i. In contrast, decoder ψ_θ tries to make its output i' indistinguishable with i to fool D. Although this paradigm generally alleviates the obstacle of "over-smoothing" [33], it creates the new challenge of mode collapse; that is, ψ_θ generates realistic (albeit very limited) i' from any inference inputs. From a holistic perspective, the use of a discriminator mainly ensures that the reconstructed i' is near i from a *distribution* perspective; no guarantee is provided from the view of a single data point.

Privacy-Aware Indicators In addition to the two standard objective functions mentioned above, we further take into account a set of *privacy-aware indicators*. As shown in Fig. 2(c), we extend discriminator D such that it checks whether the reconstructed outputs preserve the "privacy" in an explicit manner. Table 1 lists the privacy indicators used in our framework, which correspond to exploited media data of different types. For instance, for face photos (CelebA), we specify checking the identity. Hence, the enhanced discriminator D serves as a classifier to check whether the identity of the person is preserved, which thus forces decoder ψ_θ to decode the identity information in the zero-sum game. A specific fully connected layer is appended to the discriminator D in accordance with each privacy indicator.

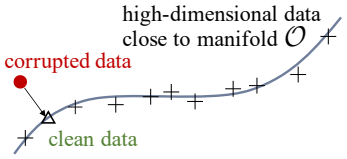

Figure 4: Denoising corrupted data during manifold learning.

Noisy Side Channel. Reconstructing media data from noisy side channels is of particular importance, because adversaries often face considerable noise in real-world attack scenarios. Manifold learning features denoising by design, the schematic view of which is presented in Fig. 4. Overall, manifold learning forces side channel traces O to concentrate near the learned low-dimensional manifold $\mathcal{O}$, where a corrupted high-dimensional data point $\tilde{o}$ (● in Fig. 4) should typically remain *orthogonal* to the manifold $\mathcal{O}$ [3]. Thus, when the decoder ψ_θ learns to reconstruct media data $i \in I$ from the representations lying on the joint manifold, corrupted $\tilde{o}$ can be fixed by first being projected onto the Δ in the manifold for denoising; and i can then be reconstructed from the Δ [18].

4.2 Fault Localization with Neural Attention

Some studies have detected software vulnerabilities that lead to side channel attacks [4, 9, 38, 39]. However, we note that such studies typically use heavyweight program-analysis techniques, such as abstract interpretation, symbolic execution, and constraint solving. Thus, performing scalable program analysis of real-world media software could prove a great challenge, given that such media software usually contains complex program structures (e.g., nested loops) and a large code base. Furthermore, the primary focus of previous studies has been crypto libraries (e.g., OpenSSL [31]), whose "sensitive data" are private key bytes or random numbers. In contrast, modeling potentially lengthy media data with various strictly defined formats could impose a further challenge (e.g., symbolizing such complex input formats) that may require the incorporation of domain-specific knowledge.

Inspired by advances in program neural smoothing [34, 35] and SCA based on neural networks [20, 32, 47], we seek to overcome question "which program point leaks side channel information" by answering the following question:

> "Which records on a logged side channel trace contribute most to the reconstruction of media data?"

Although answering the former question often requires rigorous and unscalable static analysis, the second question can be addressed smoothly by extending the encoder ϕ_θ with *attention* [43], a well-established mechanism that improves the representation of interest by telling the neural network where and upon what to focus. In particular, by enhancing the autoencoder with attention, our framework *automatically* flags side channel logs that make a primary contribution to input reconstruction. These logs are *automatically* mapped to the

corresponding memory access instructions. We can then *manually* identify the corresponding "buggy" source code. For the last step, our current experiments rely on symbol information in the assembly programs to first identify corresponding functions in source code, and then narrow down to code fragments inducing input leakage.

Despite attention being a standard mechanism to boost deep learning models [37, 43], attention in our new scenario acts like a "bug detector" to principally ease localizing vulnerable program points. In contrast to program analysis-based approaches [4, 9, 38, 39], our solution is highly scalable and incurs no extra cost during exploitation. Moreover, it analyzes software in a black-box setting that is agnostic to media software implementation details or input formats.

Fig. 2(a) depicts the enhanced trace encoder with attention. An attention module (we follow the design in [43] given its simplicity and efficiency) is inserted within every two stacked CNN layers. Let the intermediate input of a CNN layer as $C \times H \times W$, the "Channel-Attention" module $A_{channel}$ processes each segment of $1 \times H \times W$ data points from C channels and tells the encoder "where" to focus on by assigning different weights to each segment. The "Spatial-Attention" module $A_{spatial}$ processes each segment of $C \times 1 \times 1$ records and advises the encoder "what to locate" by assigning different weight on each record. From a holistic perspective, attention module $A_{channel}$ projects a coarse-grained focus on potentially interesting segments, while $A_{spatial}$ further identifies interesting side channel records in a segment.[3]

4.3 Mitigation with Perception Blinding

This section presents mitigation against manifold learning–enabled SCA (Sec. 4.1). Consistent with our attack and fault-localization, mitigation is also agnostic to particular media software and input types. We only need to perturb the media input I with pre-defined perception blinding masks.

We first introduce blinding images of the human face, and then explain how to extend perception blinding toward other input types. We also refer readers to our extended paper [2] for visualization of the perception blinding workflow and further clarification on the application scope.

A Working Example. As introduced in Sec. 2, manifold learning casts images of the human face into a set of perceptually meaningful representations; typical representations include hair style, age, and skin color. Hence, we define a *universal* mask i_{mask} of human face, such that by perturbing arbitrary images i of human face with i_{mask}, the produced output $i_{blinded}$ will be primarily projected to the same intrinsic coordinates z_{mask} in the manifold space $\mathcal{M}$. To use perception blinding, users only need to pick *one* mask i_{mask} to blind all input images i. Consequently, adversaries are restricted to the generation of media data perceptually correlated to z_{mask}. Particularly, to perturb i, we add i_{mask} as follows:

$$i_{blinded} = \alpha \times i \oplus \beta \times i_{mask}$$

where we require $\beta \gg \alpha$ and $\alpha + \beta = 1$. Perceptual contents of i_{mask} thus "dominates" the projected low-dimensional perceptions in $\mathcal{M}$. Let $P(i_{blinded})$ be the output of media software after processing $i_{blinded}$, and the user can recover the desired output by subtracting $P(i_{mask})$ from the output as follows:

$$P(i_{private}) = \frac{1}{\alpha} \times (P(i_{blinded}) \ominus \beta \times P(i_{mask}))$$

where $P(i_{private})$ is the desired output, and $P(i_{mask})$ can be pre-computed. $\oplus$ and $\ominus$ directly operate $i \in I$ of various formats, as will be defined later in this section. Because typical operations of media software (e.g., compression) are *independent* of the perceptual meaning of media inputs, the proposed blinding scheme introduces no extra hurdle for media software. In contrast, as shown in [1], SCA based on manifold learning can be mitigated in a highly effective manner.

Requirement of i_{mask}. Comparable to how RSA blinding is used to mitigate timing channels [5], perception blinding is specifically designed to mitigate manifold learning-based SCA. We require that i_{mask} must lie in the same low-dimensional manifold with the private data. Thus, i_{mask} must manifest high **perception correlation** with media software inputs $i_{private} \in I$. This shall generally ensure two properties: 1) the privacy (in terms of certain perceptions, such as gender and skin color) in $i_{private}$ can be successfully "covered" by i_{mask}, and 2) i_{mask} imposes nearly no information loss on recovering $P(i_{private})$ from $P(i_{blinded})$ except a mild computational cost due to mask operations. Considering Fig. 4, when violating this requirement of *perception correlation*, for instance, such as by using random noise to craft i_{mask}, the intrinsic coordinate of the original input (Δ) can likely drift to a "corrupted input" ($\bullet$) that is mostly orthogonal to the manifold of I. As explained in Sec. 4.1, due to the inherent noise resilience of manifold learning, crafting such a corrupted input can cause less challenge to attackers when recovering i from the low-level manifold space. Although $i_{private}$ is of low weight in $i_{blinded}$, it can still be reconstructed to some extent; results are given in the extended version [1].

Extension to other data types. For image and audio data, we recommend using a normal image $i \in I$ as the mask i_{mask}. Intuitively, by amplifying i_{mask} with a large coefficient β in generating $i_{blinded}$, i_{mask} is presumed to dominate the perceptual features in $i_{private}$. Hence, we stealthily hide the private perceptual features of $i_{private}$ in $i_{blinded}$, whose contents are difficult for adversaries to disentangle without knowing i_{mask}. For textual data, we recommend inserting notional words of high frequency to blind $i_{private}$. We present empirical results on how various choices of i_{mask} can influence the mitigation effectiveness in [1].

Implementation of Operators $\oplus$ And $\ominus$. For image and audio data, we use floating-point number addition and subtraction to implement $\oplus$ and $\ominus$. Textual data are discrete:

[3] It is well accepted that a CNN is organized in the form of `num_channels` × `width` × `height`. Therefore, we name two attention components as $A_{channel}$ and $A_{spatial}$, which are aligned with the convention.

Table 2: Statistics of side channel traces and media software. There is *no* overlapping between training and testing data.

Dataset	Information	Training Split	Testing Split	Trace Length	Matrix Encoding	Media Software
CelebA [27]	Large-scale celebrity face photos	162,770	19,962	$338,123 \pm 14,264$	$6 \times 256 \times 256$	`libjpeg` (ver. 2.0.6)
ChestX-ray [40]	Hospital-scale chest X-ray images	86,524	25,596	$329,155 \pm 10,186$	$6 \times 256 \times 256$	LOC: 103,273
SC09 [41]	Human voice of saying number 0–9	18,620	2,552	$1,835,067 \pm 103,328$	$8 \times 512 \times 512$	`ffmpeg` (ver. 4.3)
Sub-URMP [23]	Sound clips of 13 instruments	71,230	9,575	$1,678,485 \pm 36,122$	$8 \times 512 \times 512$	LOC: 1,236,079
COCO [26]	Image captions	414,113	202,654	$77,796 \pm 14$	$6 \times 128 \times 128$	`hunspell` (vers. 1.7.0)
DailyDialog [24]	Sentences of daily chats	11,118	1,000	$77,799 \pm 102$	$6 \times 128 \times 128$	LOC: 39,096

Reconstructed Text	Reference Input
*I think it **' be better for find** a good babysitter here . It **' be cost , an** or three days .*	*I think it would be better to have a good babysitter here . It might even be for two or three days .*
*She **is** a **single** cold , and **it** don ' t want to take **care to** us . But we don ' t like **how** can stay with our .*	*She has a bad cold , and we don ' t want to take her with us . But we don ' t know who can stay with her .*
*I ' m sorry **, say that** . What ' s wrong with her ?*	*I ' m sorry to hear it . What ' s wrong with her ?*

Figure 5: Reconstructed sentences from logged cache line accesses. We mark ***inconsistent reconstructions***.

considering that media software often manipulates textual data at the word level, simply "adding" or altering words in the input text will likely trigger some error handling routines of the corresponding media software, which is not desirable. Sec. 4.1 clarifies that our autoencoder framework essentially captures the "word dependency" between words in a sentence; accordingly, we define the $\oplus$ operation as inserting words in a sentence, whereas the $\ominus$ operation is implemented to remove previously inserted words.

5 Results

Media Software and Media Dataset. Table 2 reports evaluated media software and statistics of side channel traces. All media software are complex real-world software, e.g., `FFmpeg` contains 1M LOC. We prepare two common datasets for each media software to comprehensively evaluate our attack. All datasets contain daily media data that, once exposed to adversaries, would result in privacy leakage. We compile all three media software into 64-bit binary code using `gcc` on a 64-bit Ubuntu 18.04 machine.

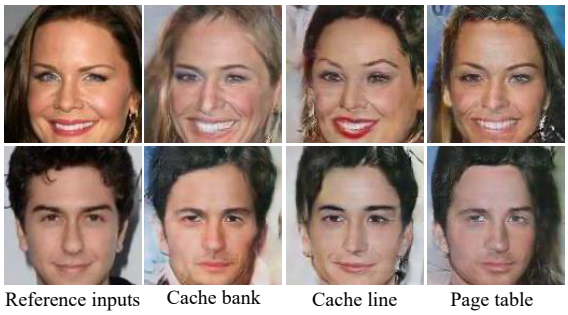

Figure 6: Reconstructed face photos.

Case Studies. Fig. 6 and Fig. 7 show face photos reconstructed using different side channels. We can observe highly correlated visual appearances, including gender, face orientation, skin color, nose shape, and hair styles. Fig. 8 shows reconstructed Chest X-ray photos and Fig. 5 shows reconstructed text. For reconstructed audios, please see [2].

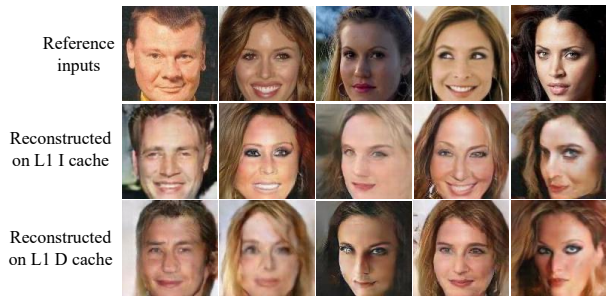

Figure 7: Reconstructed images using cache set access logged on L1 cache (via `Prime+Probe`) of Intel Xeon CPU.

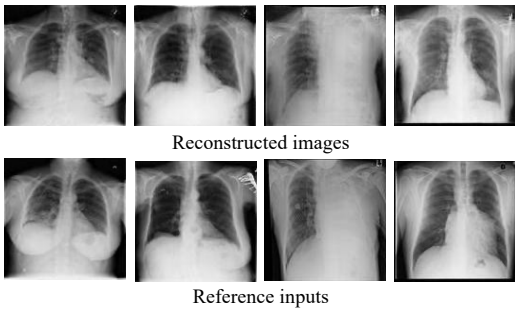

Figure 8: Reconstructed Chest X-ray images from cache line.

References

[1] Extended version. https://arxiv.org/pdf/2112.04947.pdf.

[2] Research artifact. https://github.com/Yuanyuan-Yuan/Manifold-SCA.

[3] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

[4] Robert Brotzman, Shen Liu, Danfeng Zhang, Gang Tan, and Mahmut Kandemir. CaSym: Cache aware symbolic execution for side channel detection and mitigation. In *IEEE SP*, 2018.

[5] David Brumley and Dan Boneh. Remote timing attacks are practical. *Computer Networks*, 48(5):701–716, 2005.

[6] Mikhail Chernov and Eric Ghysels. A study towards a unified approach to the joint estimation of objective and risk neutral measures for the purpose of options valuation. *Journal of financial economics*, 56(3):407–458, 2000.

[7] Jose A Costa and Alfred O Hero. Geodesic entropic graphs for dimension and entropy estimation in manifold learning. *IEEE Transactions on Signal Processing*, 52(8):2210–2221, 2004.

[8] Craig Disselkoen, David Kohlbrenner, Leo Porter, and Dean Tullsen. Prime+Abort: A timer-free high-precision L3 cache attack using Intel TSX. In *USENIX Sec.*, 2017.

[9] Goran Doychev, Dominik Feld, Boris Kopf, Laurent Mauborgne, and Jan Reineke. CacheAudit: A tool for the static analysis of cache side channels. In *USENIX Sec.*, 2013.

[10] Goran Doychev and Boris Köpf. Rigorous analysis of software countermeasures against cache attacks. PLDI, 2017.

[11] Xing Fan, Wei Jiang, Hao Luo, and Mengjuan Fei. Spherereid: Deep hypersphere manifold embedding for person re-identification. *Journal of Visual Communication and Image Representation*, 60:51–58, 2019.

[12] Marcus Hähnel, Weidong Cui, and Marcus Peinado. High-resolution side channels for untrusted operating systems. USENIX ATC, 2017.

[13] Xiaofei He, Shuicheng Yan, Yuxiao Hu, Partha Niyogi, and Hong-Jiang Zhang. Face recognition using laplacianfaces. *IEEE transactions on pattern analysis and machine intelligence*, 27(3):328–340, 2005.

[14] Benjamin Hettwer, Stefan Gehrer, and Tim Güneysu. Profiled power analysis attacks using convolutional neural networks with domain knowledge. SAC, 2018.

[15] Benjamin Hettwer, Tobias Horn, Stefan Gehrer, and Tim Güneysu. Encoding power traces as images for efficient side-channel analysis. *arXiv preprint arXiv:2004.11015*, 2020.

[16] Annelie Heuser and Michael Zohner. Intelligent machine homicide. In *COSADE*, 2012.

[17] Geoffrey E Hinton and Richard S Zemel. Autoencoders, minimum description length, and helmholtz free energy. *Advances in neural information processing systems*, 6:3–10, 1994.

[18] Daniel Holden, Jun Saito, Taku Komura, and Thomas Joyce. Learning motion manifolds with convolutional autoencoders. In *SIGGRAPH Asia 2015 Technical Briefs*, pages 1–4. 2015.

[19] Gorka Irazoqui, Kai Cong, Xiaofei Guo, Hareesh Khattri, Arun K. Kanuparthi, Thomas Eisenbarth, and Berk Sunar. Did we learn from LLC side channel attacks? A cache leakage detection tool for crypto libraries. *CoRR*, 2017.

[20] Jaehun Kim, Stjepan Picek, Annelie Heuser, Shivam Bhasin, and Alan Hanjalic. Make some noise. unleashing the power of convolutional neural networks for profiled side-channel analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 148–179, 2019.

[21] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[22] John A Lee and Michel Verleysen. *Nonlinear dimensionality reduction*. Springer Science & Business Media, 2007.

[23] Bochen Li, Xinzhao Liu, Karthik Dinesh, Zhiyao Duan, and Gaurav Sharma. Creating a multitrack classical music performance dataset for multimodal music analysis: Challenges, insights, and applications. *IEEE Transactions on Multimedia*, 21(2):522–535, 2018.

[24] Yanran Li, Hui Su, Xiaoyu Shen, Wenjie Li, Ziqiang Cao, and Shuzi Niu. Dailydialog: A manually labelled multi-turn dialogue dataset. *arXiv preprint arXiv:1710.03957*, 2017.

[25] Tong Lin and Hongbin Zha. Riemannian manifold learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(5):796–809, 2008.

[26] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft COCO: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[27] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. ICCV, 2015.

[28] Chi-Keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff Lowney, Steven Wallace, Vijay Janapa Reddi, and Kim Hazelwood. Pin: building customized program analysis tools with dynamic instrumentation. In *Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation (PLDI'05)*, 2005.

[29] Houssem Maghrebi, Thibault Portigliatti, and Emmanuel Prouff. Breaking cryptographic implementations using deep learning techniques. In *International Conference on Security, Privacy, and Applied Cryptography Engineering*, pages 3–26. Springer, 2016.

[30] Francisco J Martinez-Murcia, Andres Ortiz, Juan-Manuel Gorriz, Javier Ramirez, and Diego Castillo-

Barnes. Studying the manifold structure of alzheimer's disease: A deep learning approach using convolutional autoencoders. *J-BHI*, 24(1):17–26, 2019.

[31] Openssl. https://www.openssl.org/.

[32] Stjepan Picek, Ioannis Petros Samiotis, Jaehun Kim, Annelie Heuser, Shivam Bhasin, and Axel Legay. On the performance of convolutional neural networks for side-channel analysis. In *International Conference on Security, Privacy, and Applied Cryptography Engineering*, pages 157–176. Springer, 2018.

[33] Yuki Saito, Shinnosuke Takamichi, and Hiroshi Saruwatari. Statistical parametric speech synthesis incorporating generative adversarial networks. *TASLP*, 26(1):84–96, 2017.

[34] Dongdong She, Yizheng Chen, Abhishek Shah, Baishakhi Ray, and Suman Jana. Neutaint: Efficient dynamic taint analysis with neural networks. IEEE SP, 2020.

[35] Dongdong She, Kexin Pei, Dave Epstein, Junfeng Yang, Baishakhi Ray, and Suman Jana. NEUZZ: Efficient fuzzing with neural program smoothing. IEEE SP, 2019.

[36] Eran Tromer, DagArne Osvik, and Adi Shamir. Efficient cache attacks on AES, and countermeasures. *Journal of Cryptology*, 23(1):37–71, 2010.

[37] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

[38] Shuai Wang, Yuyan Bao, Xiao Liu, Pei Wang, Danfeng Zhang, and Dinghao Wu. Identifying cache-based side channels through secret-augmented abstract interpretation. USENIX Security, 2019.

[39] Shuai Wang, Pei Wang, Xiao Liu, Danfeng Zhang, and Dinghao Wu. CacheD: Identifying cache-based timing channels in production software. In *26th USENIX Security Symposium*, pages 235–252, 2017.

[40] Xiaosong Wang, Yifan Peng, Le Lu, Zhiyong Lu, Mohammadhadi Bagheri, and Ronald M Summers. Chestx-ray8: Hospital-scale chest x-ray database and benchmarks on weakly-supervised classification and localization of common thorax diseases. CVPR, 2017.

[41] Pete Warden. Speech commands: A dataset for limited-vocabulary speech recognition. *arXiv preprint arXiv:1804.03209*, 2018.

[42] Jan Wichelmann, Ahmad Moghimi, Thomas Eisenbarth, and Berk Sunar. MicroWalk: A framework for finding side channels in binaries. In *ACSAC*, 2018.

[43] Sanghyun Woo, Jongchan Park, Joon-Young Lee, and In So Kweon. Cbam: Convolutional block attention module. ECCV, 2018.

[44] Yuanzhong Xu, Weidong Cui, and Marcus Peinado. Controlled-channel attacks: Deterministic side channels for untrusted operating systems. IEEE SP, 2015.

[45] Yuval Yarom. Mastik: A micro-architectural side-channel toolkit. *Retrieved from School of Computer Science Adelaide: http://cs.adelaide.edu.au/yval/Mastik*, 16, 2016.

[46] Yuval Yarom, Daniel Genkin, and Nadia Heninger. Cachebleed: a timing attack on openssl constant-time rsa. *Journal of Cryptographic Engineering*, 7(2):99–112, 2017.

[47] Yuanyuan Yuan, Shuai Wang, and Junping Zhang. Private image reconstruction from system side channels using generative models. In *ICLR*, 2021.

[48] Yinqian Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. Cross-VM side channels and their use to extract private keys. CCS, 2012.

[49] Bo Zhu, Jeremiah Z Liu, Stephen F Cauley, Bruce R Rosen, and Matthew S Rosen. Image reconstruction by domain-transform manifold learning. *Nature*, 555(7697):487–492, 2018.

[50] Jun-Yan Zhu, Philipp Krähenbühl, Eli Shechtman, and Alexei A Efros. Generative visual manipulation on the natural image manifold. In *European conference on computer vision*, pages 597–613. Springer, 2016.