“*I Know What You See Up There*”: Inferring Media Software Inputs via Side Channel Leakage with Manifold Learning

Yuanyuan Yuan[1], Qi Pang[1], Zhibo Liu[1], Tianxiang Li[2], Wenqiang Li[2], Shuai Wang[1]

[1]The Hong Kong University of Science and Technology, [2]Independent Researcher

1 Threat Model and Attack Scenarios

This study reconstructs confidential inputs (e.g., image, audio, and text) 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 side channel analysis (SCA) [11–13, 17, 24] 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 [6, 10, 37, 39], and this assumption has been consistently made by many previous works [4, 8, 16, 31, 32, 35]. In this study, we use Intel Pin [23] 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 [29] to log cache activities when media software is processing a secret input. We use Mastik [38], 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 [10, 37] or on crypto libraries [41], 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. 4 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, 7, 32]. Direct reconstruction of media data is beyond the scope of such formal method-based techniques, and these studies did not propose SCA mitigation.

2 A Manifold View on SCA of Media Software

The use of manifold underlies the feasibility of dimensionality reduction [18]. 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 [9]. 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, 42, 43].

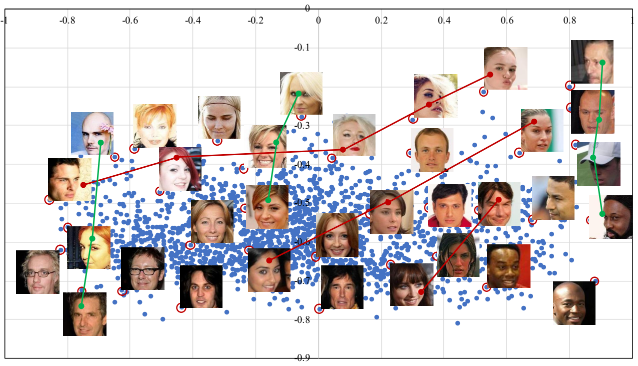

Figure 1: Project face photos to a 2-dimensional manifold.

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 framework (see Sec. 3) 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 [43]. 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 [43].

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) [10, 37]; 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 [42, 43] led us to construct a joint, *perception-level* connection between side channel logs and high-dimensional media inputs.[1] 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 [42].

3 Approach

3.1 SCA with Autoencoder

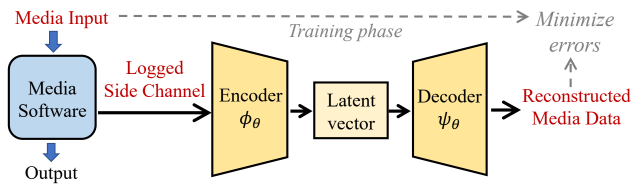

Figure 2: Reconstructing media input using an autoencoder.

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 [14] defines a parametric feature-extracting function ϕ_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.

Encoder ϕ_θ. A logged side channel trace will first be folded into a $K \times N \times N$ matrix (see Table 1 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 [18], 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).

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_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_θ. Accordingly, we design different ψ_θ for continuous and discrete media data; see details in [1].

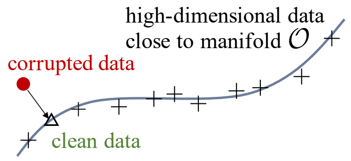

Figure 3: Denoising corrupted data during manifold learning.

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. 3. 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. 3) 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 [15].

3.2 Fault Localization with Neural Attention

Some studies have detected software vulnerabilities that lead to side channel attacks [4, 7, 31, 32]. 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

[1]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.

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 [25]), 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 [27,28] and SCA based on neural networks [17,26,40], 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* [36], 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 [30,36], attention in our new scenario acts like a "bug detector" to principally ease localizing vulnerable program points. In contrast to program analysis-based approaches [4,7,31,32], 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.

3.3 Mitigation with Perception Blinding

This section presents mitigation against manifold learning–enabled SCA (Sec. 3.1). 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. 1, 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. 3, 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" (●) that is mostly orthogonal to the manifold of I. As explained in Sec. 3.1, 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}.

Table 1: 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 [22]	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 [33]	Hospital-scale chest X-ray images	86,524	25,596	$329,155 \pm 10,186$	$6 \times 256 \times 256$	LOC: 103,273
SC09 [34]	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 [19]	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 [21]	Image captions	414,113	202,654	$77,796 \pm 14$	$6 \times 128 \times 128$	`hunspell` (vers. 1.7.0)
DailyDialog [20]	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 4: Reconstructed sentences from logged cache line accesses. We mark ***inconsistent reconstructions***.

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: 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. 3.1 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.

4 Results

Media Software and Media Dataset. Table 1 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.

Case Studies. Fig. 5 and Fig. 6 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. 7 shows reconstructed Chest X-ray photos and Fig. 4 shows reconstructed text. For reconstructed audios, please see [2].

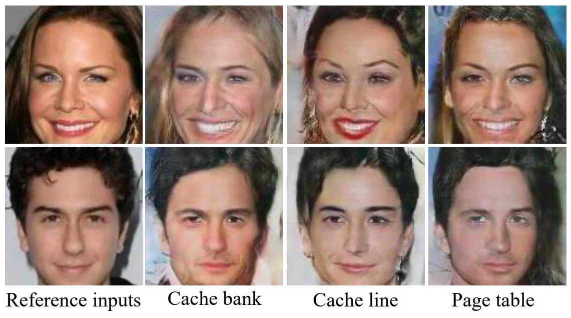

Figure 5: Reconstructed face photos.

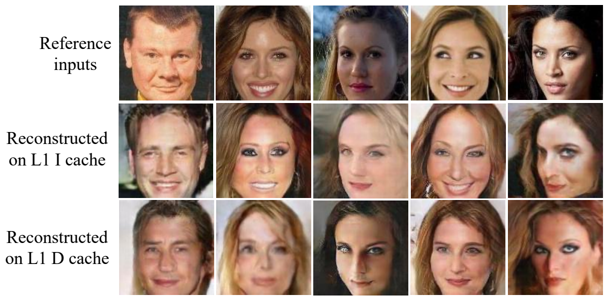

Figure 6: Reconstructed images using cache set access logged on L1 cache (via `Prime+Probe`) of Intel Xeon CPU.

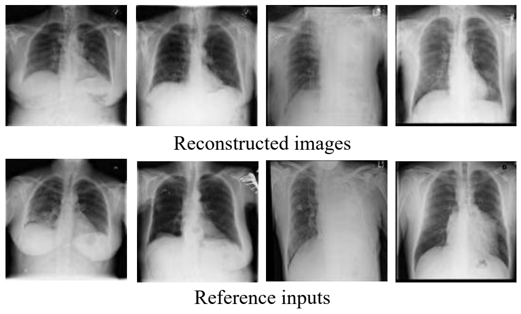

Figure 7: Reconstructed Chest X-ray images from cache line.

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] 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.

[7] 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.

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

[9] 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.

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

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

[12] 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.

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

[14] 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.

[15] 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.

[16] 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.

[17] 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.

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

[19] 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.

[20] 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.

[21] 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.

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

[23] 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.

[24] 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.

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

[26] 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.

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

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

[29] Eran Tromer, DagArne Osvik, and Adi Shamir. Efficient

cache attacks on AES, and countermeasures. *Journal of Cryptology*, 23(1):37–71, 2010.

[30] 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.

[31] 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.

[32] 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.

[33] 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.

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

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

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

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

[38] 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.

[39] 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.

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

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

[42] 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.

[43] 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.