USENIX Security '25 Artifact Appendix: A Crack in the Bark: Leveraging Public Knowledge to Remove Tree-Ring Watermarks

Junhua Lin Marc Juarez
University of Edinburgh

A Artifact Appendix

A.1 Abstract

We provide the source code used in our experiments which implements the set of attacks that successfully remove a Tree-Ring Watermark. The primary contribution is an implementation of projected gradient descent using a surrogate classifier which operates in the latent space of the diffusion model. Currently our attacks only operate within the latent space provided by Stable Diffusion v2.1, SDXL and Ostris (a 16-Channel VAE). We also provide the implementation of another attack called Adversarial noising, created by Nils et al., which was adapted to our evaluation framework. Additionally, we provide the images used as the target class images which our attacks attempt to imitate. This is given as two sets, one set (named wmvsunwm_imagenet) contains 1500 images to be used as reference images in assessing Wm vs Unwm attacks and the other set (named wmvspub_imagenet) contains 7000 images to be used in Wm vs Public attacks. Both sets of images have their corresponding set of prompts inside the prompts directory of the repository.

A.2 Description & Requirements

Within the repository, we include a README file to describe the Python environment used to produce the results reported in our paper. Additionally, we also provide a DockerFile to allow users to build a virtual machine with all the necessary libraries. Beyond the libraries we also require the user to install model weights that need to be placed within the base directory of the repository. These are OpenAI's imagenet diffusion weights [1] and Alexnet weights [2].

A.2.1 Security, privacy, and ethical concerns

Our attacks specifically target images and does not affect the system in any way. Furthermore, the datasets and models used are open sourced by their respective authors. As such, there is no risk associated with the security of the machine or data privacy. With regard to ethical concerns, Tree-Ring watermarking is still in the prototype stage and has not been deployed. Therefore, the likelihood of malicious actors exploiting these attacks is minimal, while exposing their vulnerabilities gives a better understanding of the limitations of ML watermarking.

A.2.2 How to access

A stable version of the artifact is provided in Zenodo [3]. The stable URL points to the latest version of the artifact which contains three archives. These are `wmvspub_imagenet.tar.gz` which contains 7000 images sampled from ImageNet, `wmvsunwm_imagenet.tar.gz` which contains 1500 images from ImageNet and `crack-in-the-bark-main.zip` which contains the source code for our experiments.

A.2.3 Hardware dependencies

Our experiments were run on an Nvidia A100 GPU with 80GB of VRAM with 8 CPU cores. The minimum memory requirement is 32GB with atleast 50GB of storage available.

A.2.4 Software dependencies

All software dependencies are listed in the README file inside the repository. The minimum requirements are Python and a package manager (e.g. conda/pip/mamba). The Dockerfile also constructs a machine image with the Python environment already setup.

A.2.5 Benchmarks

The datasets are provided within the Zenodo repository for this project. The diffusion models such as Stable Diffusion, SDXL's VAE of Ostris's VAE are installed when the code is run. Additional models such as Alexnet's weights and Guided Diffusion's weights were detailed in Section A.2.

[1] https://openaipublic.blob.core.windows.net/diffusion/jul-2021/512x512_diffusion.pt

[2] https://github.com/richzhang/PerceptualSimilarity/tree/master/lpips/weights

[3] https://doi.org/10.5281/zenodo.15595719

A.3 Set-up

A.3.1 Installation

Installation of packages should be done using a python package manager (e.g. conda/pip/mamba) if not using the Docker image. The version of Python we use is 3.10 and the CUDA drivers are version 12.6. After unzipping the source code, you will need to install and unzip the imagenet datasets from Zenodo into the base directory of the repository. You will also need to clone the Perceptual Similarity repository and move the "weights" directory inside the "lpips" directory to the base of the repository. Finally, you will need to download the Guided Diffusion weights from OpenAI and place them within the base of the repository.

Note that the version of `huggingface_hub` used has a bug related to the version of the `diffusers` library used. As such, we require the user to manually edit the `dynamic_modules_utils.py` file. Details on what to edit and where to find the file are described in the source code's README under the section 'Dependencies'. This step can be skipped if

A.3.2 Basic Test

To verify that the environment has been properly set up, we provide a simple verification experiment via 'verify.sh'. Running this script will start a pipeline which goes through each stage of the attack. This simple experiment should take no longer than 10 minutes, depending on the GPU used. After this script has finished running, a log directory will be made containing two folders, one containing all the data, i.e. the cached latents, images, prompt to image tables e.t.c. The other containing the evaluation results. Additionally, an output folder should have been created with two subdirectories, 'images' and 'models'. The should contain the output images of the attack and the surrogate model used to execute the attack respectively.

A.4 Evaluation workflow

A.4.1 Major Claims

The main claims made in our paper are:

(C1): All the surrogate detectors (trained on Raw Pixel Values, True Latent Vectors, and VAE-Recovered Latent Vectors) achieve high accuracy and none of them significantly overfits. This is shown by experiments (E3)–(E6), and (E11) and (E12) described in Section 5.2 whose results are reported in Table 1.

(C2): Our evaluation of adversarial noising achieves a TPR@1%FPR of 0.141 for $\delta = 2$, slightly higher than the reported 0.052 of the original paper. This is proven by experiment (E2) described in Section 5.3 and reported in the second row of Table 2.

(C3): We found a discrepancy between An et al.'s and our results when executing the PGD attack with a surrogate trained on the Raw Pixel Values of the Wm & Pub dataset: they reported an average TPR@1%FPR of 0.99, but our results show a value of 0.48. This is proven by experiments (E3) and (E4) described in Section 5.3 and reported in the third row of Table 2.

(C4): Our PGD attack on the True Latents Vectors (fourth row of Table 2) performs worse than our surrogate attack trained on VAE-Recovered Latent Vectors. This is supported by experiments (E5), (E6), (E11), and (E12) described in Section 5.3 and reported in rows 5 and 6 of Table 2.

(C5): Among the attacks we evaluated, our proposed attack by recovering latents using AutoencoderKL (the same autoencoder as the target) results in the largest drop in accuracy, while maintaining image quality comparable to the baseline. This is supported by experiments (E2)–(E6), and (E11) and (E12), described in Section 5.3 and reported in Table 2.

(C6: All the attacks included in our evaluation are significantly more effective than any of the approaches tested by Wen et al.: they report an average ROC-AUC of 0.975 and TPR@1%FPR of 0.694, while the attacks we evaluate achieve an average of 0.519 ROC-AUC and 0.261 TPR@1%FPR, representing a 1.9x decrease in overall detection accuracy. Our results are obtained by conducting experiments (E2)–(E6), and (E11) and (E12), described in Section 5.3 and reported in Table 2.

(C7): Using SDXL's VAE, the attack remains robust, though less effective than when using the victim model's VAE and with lower image quality. This is supported by experiments (E7) and (E8) described in Section 5.4 and reported in row 5 of Table 2.

(C8): Training the attack with the SDXL's VAE on the Wm & UnWm dataset significantly degrades image quality, with the FID score increasing by 33.71 and similar increases in LPIPS and CLIP Score. This is demonstrated by experiments (E7) and (E8) described in Section 5.4 and shown in row 5 of Table 2.

(C9): The 16-Channel VAE's performance is comparable to training the surrogate detector on raw images. This is shown by experiments (E9) and (E10) described in Section 5.4 and shown in row 6 of Table 2.

(C10): While the attack still decreases the effectiveness of Tree-Ring, it is not as successful as when the target VAE is available. This is supported by experiments (E13) and (E14) described in Section 5.5 and reported in Table 3.

(C11): In the no-attack setting, for base rates below 0.1, precision decreases significantly, even for high TPRs. This is supported by experiments (E1) and (E15) described in Section 5.6 and illustrated in the no-attack subplot of Figure 8.

A.4.2 Experiments

Within the repository, a detailed README is provided, called ‘example_pipelines.md‘. After the setup instructions provided in Section A.3.1, all thats needed is to follow the example pipelines provided whilst substituting the names of directories automatically generated after completion of certain scripts.

(E1): No Attack Baseline [20 human-minutes + 12 compute-hours + 1GB disk]: Generate watermarked images and assess detection performance without any attack to establish baseline metrics.
How to: Follow the "No Attack" pipeline in example_pipelines.md. Run generate_data.py then assess_images.py using the same watermarked images as both original and adversarial inputs to establish baseline detection rates.
Results: Baseline watermark detection performance metrics (PRC-AUC, ROC-AUC, Accuracy, TPR@1%FPR) and image quality scores (LPIPS, CLIP, FID) stored in summary.json under the assessment logs directory. If no changes were made to the --run_name parameter, the directory created will have "no_attack" in its name.

(E2): Adversarial Noising Attack [20 human-minutes + 24 compute-hours + 2GB disk]: Reproduce the baseline adversarial noising attack from prior work for comparison with our methods.
How to: Follow the "Adversarial Noising" pipeline in example_pipelines.md. Generate watermarked data, then run remove_watermark.py with --attack=adv_noising and --mode=rawpix, followed by assessment.
Results: Adversarial Noising performance metrics (PRC-AUC, ROC-AUC, Accuracy, TPR@1%FPR) and image quality scores (LPIPS, CLIP, FID) stored in summary.json under the assessment logs directory. If no changes were made to the --run_name parameter, the directory created will have "adv_noising" in its name.

(E3): Raw Pixel Surrogate Attack - WM vs UnWM [20 human-minutes + 8 compute-hours + 2GB disk]: Train surrogate model on raw pixels comparing watermarked vs generated non-watermarked images, then perform PGD-based removal.
How to: Follow the "Raw Pixel Values - Wm vs UnWm" pipeline in example_pipelines.md. Generate both watermarked and non-watermarked images, train surrogate with --mode=rawpix, then perform removal and assessment.
Results: Raw Pixel Wm vs UnWm performance metrics (PRC-AUC, ROC-AUC, Accuracy, TPR@1%FPR) and image quality scores (LPIPS, CLIP, FID) stored in summary.json under the assessment logs directory. If no changes were made to the --run_name parameter, the directory created will have "wmvsunwm_rawpix" in its name.

(E4): Raw Pixel Surrogate Attack - WM vs Public [20 human-minutes + 16 compute-hours + 8GB disk]: Train surrogate model comparing watermarked images vs public dataset images for more realistic attack scenario.
How to: Follow the "Raw Pixel Values - Wm vs Public" pipeline in example_pipelines.md. Generate watermarked images, train surrogate against wmvspub_imagenet dataset with --mode=rawpix, then perform removal and assessment.
Results: Raw Pixel Wm vs Public performance metrics (PRC-AUC, ROC-AUC, Accuracy, TPR@1%FPR) and image quality scores (LPIPS, CLIP, FID) stored in summary.json under the assessment logs directory. If no changes were made to the --run_name parameter, the directory created will have "wmvspub_rawpix" in its name.

(E5): True Latent Vector Attack - WM vs UnWM [20 human-minutes + 8 compute-hours + 2GB disk]: Attack using true latent vectors from diffusion process comparing watermarked images vs generated non-watermarked images.
How to: Follow the "True Latent Vectors - Wm vs UnWm" pipeline in example_pipelines.md. Generate data with --save_raw_latent flag when running generate_data.py, train surrogate, perform latent-space removal and assess.
Results: True Latent Wm vs UnWm performance metrics (PRC-AUC, ROC-AUC, Accuracy, TPR@1%FPR) and image quality scores (LPIPS, CLIP, FID) stored in summary.json under the assessment logs directory. If no changes were made to the --run_name parameter, the directory created will have "wmvsunwm_true_latent" in its name.

(E6): True Latent Vector Attack - WM vs Public [20 human-minutes + 16 compute-hours + 8GB disk]: Attack using true latent vectors from diffusion process comparing watermarked images vs public images.
How to: Follow the "True Latent Vectors - Wm vs Public" pipeline in example_pipelines.md. Generate watermarked data with --save_raw_latent, recover latents for public images using recover_latents.py, train, attack and assess.
Results: True Latent Wm vs Public performance metrics (PRC-AUC, ROC-AUC, Accuracy, TPR@1%FPR) and image quality scores (LPIPS, CLIP, FID) stored in summary.json under the assessment logs directory. If no changes were made to the --run_name parameter, the directory created will have "wmvspub_true_latent" in its name.

(E7): SDXL-VAE Latent Attack - WM vs UnWM [20 human-minutes + 8 compute-hours + 2GB disk]: Attack using SDXL VAE-recovered latents against generated

non-watermarked dataset.

How to: Follow the "SDXL-VAE Latent Vectors - Wm vs UnWm" pipeline in `example_pipelines.md`. Generate images, recover latents using SDXL VAE (`stabilityai/sdxl-vae`), train surrogate, perform removal and assess images

Results: SDXL-VAE Wm vs UnWm performance metrics (PRC-AUC, ROC-AUC, Accuracy, TPR@1%FPR) and image quality scores (LPIPS, CLIP, FID) stored in `summary.json` under the assessment logs directory. If no changes were made to the `--run_name` parameter, the directory created will have "wmvsunwm_sdxl_latent" in its name.

(E8): SDXL-VAE Latent Attack - WM vs Public [20 human-minutes + 16 compute-hours + 8GB disk]: Attack using SDXL VAE-recovered latents against public dataset.

How to: Follow the "SDXL-VAE Latent Vectors - Wm vs Public" pipeline in `example_pipelines.md`. Generate watermarked images, recover SDXL latents for both watermarked and public images, train surrogate and perform attack.

Results: SDXL-VAE Wm vs Public performance metrics (PRC-AUC, ROC-AUC, Accuracy, TPR@1%FPR) and image quality scores (LPIPS, CLIP, FID) stored in `summary.json` under the assessment logs directory. If no changes were made to the `--run_name` parameter, the directory created will have "wmvsunwm_sdxl_latent" in its name.

(E9): 16-Channel VAE Attack - WM vs UnWM [20 human-minutes + 8 compute-hours + 2GB disk]: Attack using high-dimensional VAE latents with 16 channels against generated non-watermarked dataset.

How to: Follow the "16-Channel VAE Latent Vectors - Wm vs UnWm" pipeline in `example_pipelines.md`. Use `ostris/vae-kl-f8-d16` model for latent recovery, train surrogate in high-dimensional latent space, perform the attack and assess.

Results: 16-Channel VAE Wm vs UnWm performance metrics (PRC-AUC, ROC-AUC, Accuracy, TPR@1%FPR) and image quality scores (LPIPS, CLIP, FID) stored in `summary.json` under the assessment logs directory. If no changes were made to the `--run_name` parameter, the directory created will have "wmvsunwm_ostris_latent" in its name.

(E10): 16-Channel VAE Attack - WM vs Public [20 human-minutes + 16 compute-hours + 8GB disk]: Attack using high-dimensional VAE latents with 16 channels against public dataset.

How to: Follow the "16-Channel VAE Latent Vectors - Wm vs Public" pipeline in `example_pipelines.md`. Recover 16-channel latents for both watermarked and public images using `ostris/vae-kl-f8-d16`, then train, attack and assess.

Results: 16-Channel VAE Wm vs Public performance metrics (PRC-AUC, ROC-AUC, Accuracy, TPR@1%FPR) and image quality scores (LPIPS, CLIP, FID) stored in `summary.json` under the assessment logs directory. If no changes were made to the `--run_name` parameter, the directory created will have "wmvspub_ostris_latent" in its name.

(E11): VAE - Recovered Attack - WM vs UnWM [20 human-minutes + 8 compute-hours + 2GB disk]: Attack using SD VAE-recovered latents against generated non-watermarked dataset.

How to: Follow the "VAE-Recovered Latent Vectors - Wm vs UnWm" pipeline in `example_pipelines.md`. Use `stabilityai/stable-diffusion-2-1-base` for latent recovery, train surrogate with, perform attack and assess.

Results: SD VAE Wm vs UnWm performance metrics (PRC-AUC, ROC-AUC, Accuracy, TPR@1%FPR) and image quality scores (LPIPS, CLIP, FID) stored in `summary.json` under the assessment logs directory. If no changes were made to the `--run_name` parameter, the directory created will have "wmvsunwm_sd_latent" in its name.

(E12): VAE - Recovered Attack - WM vs Public [20 human-minutes + 16 compute-hours + 8GB disk]: Attack using SD VAE-recovered latents against public dataset.

How to: Follow the "VAE-Recovered Latent Vectors - Wm vs Public" pipeline in `example_pipelines.md`. Recover SD VAE latents for watermarked images and public dataset, train surrogate, perform attack and assess.

Results: SD VAE Wm vs Public performance metrics (PRC-AUC, ROC-AUC, Accuracy, TPR@1%FPR) and image quality scores (LPIPS, CLIP, FID) stored in `summary.json` under the assessment logs directory. If no changes were made to the `--run_name` parameter, the directory created will have "wmvspub_sd_latent" in its name.

(E13): Guided Diffusion Attack - WM vs UnWM [20 human-minutes + 30 compute-hours + 6GB disk]: Attack using OpenAI's Guided Diffusion model comparing against generated non-watermarked dataset.

How to: Follow the "Attack on Guided Diffusion Model - Wm vs UnWm" pipeline in `example_pipelines.md`. Use `--model_id=512x512_diffusion` for generation assuming you have placed the model at the base directory after installing from the Guided Diffusion repository, train surrogate, perform attack and assess.

Results: Guided Diffusion Wm vs UnWm performance metrics (PRC-AUC, ROC-AUC, Accuracy, TPR@1%FPR) and image quality scores (LPIPS, CLIP, FID) stored in `summary.json` under the assessment logs directory. If no changes were made to the `--run_name` parameter, the directory created will have "wmvsunwm_guided_diffusion" in its name.

(E14): Guided Diffusion Attack - WM vs Public [20 human-minutes + 48 compute-hours + 8GB disk]: Attack using OpenAI's Guided Diffusion model comparing against public dataset.

How to: Follow the "Attack on Guided Diffusion Model - Wm vs Public" pipeline in `example_pipelines.md`. Generate larger dataset `--max_num_images=7000` with Guided Diffusion, train against public images, perform attack, and assess attack performance.

Results: Guided Diffusion Wm vs Public performance metrics (PRC-AUC, ROC-AUC, Accuracy, TPR@1%FPR) and image quality scores (LPIPS, CLIP, FID) stored in `summary.json` under the assessment logs directory. If no changes were made to the `--run_name` parameter, the directory created will have "wmvspub_guided_diffusion" in its name.

(E15:) Precision vs Base Rate [20 human-minutes + 1GB disk]: Plotting the precision at different TPR@X%FPR cutoffs against several base rates.

How to: Follow the "Precision vs Base Rate Plot" example in `example_pipelines.md`. Call `precision_at_base_rate.py` with `--table_path` set to the path to a `metadata.csv` file after `assess_images.py` completes.

Setup: The assumption here is that you have atleast one experiment where `assess_images.py` has completed. After which, the logger object will create a csv file under the experiment results directory in the logs directory which contains the measurement metric for deciding the validity of a watermark.

Results: This will generate a single plot for an attack where we selected 4 TPR@X%FPR cut off points and plotted the precision against a set of base rates.

A.5 Version

Based on the LaTeX template for Artifact Evaluation V20231005. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/usenixsec2025/.