SMACK: Semantically Meaningful Adversarial Audio Attack

Zhiyuan Yu, Yuanhaur Chang, and Ning Zhang, *Washington University in St. Louis;* Chaowei Xiao, *Arizona State University*

https://www.usenix.org/conference/usenixsecurity23/presentation/yu-zhiyuan-smack

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 32nd USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 32nd USENIX Security Symposium.

August 9–11, 2023 • Anaheim, CA, USA

978-1-939133-37-3

Open access to the Artifact Appendices to the Proceedings of the 32nd USENIX Security Symposium is sponsored by USENIX.

USENIX'23 Artifact Appendix: SMACK: Semantically Meaningful Adversarial Audio Attack

Zhiyuan Yu, Yuanhaur Chang, Ning Zhang
Washington University in St. Louis

Chaowei Xiao
Arizona State University

A Artifact Appendix

A.1 Abstract

SMACK is an adversarial audio attack that leverages manipulation of the prosody attributes to craft adversarial speech examples. Our artifact comprises the source code, the generative model for controlling speech prosody, along with the automatic speech recognition (ASR) and speaker recognition (SR) models for attack testing. To operate the attack framework, the user needs to run the program in the command line, providing attack types (i.e., against ASR or SR system) and specifying attack targets (i.e, targeted transcription or speaker label). The expected results are the adversarial audio samples.

Given the complexity of the speech generative model involved in SMACK, a machine with a moderate CPU and a GPU of at least 8GB VRAM is recommended. Please note that run-time may vary depending on the user's hardware. We have compiled a list of required dependencies into a YML configuration file.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

The artifact provided does not contain any harmful materials that may compromise machine security or pose threats to human health. The generative model was trained on public speech corpus, and there is no privacy concern associated with the use of artifacts. We have taken the utmost care to ensure that the artifact meets all necessary safety standards and guidelines.

A.2.2 How to access

We have made the code and models available on Zenodo. The stable URL link pointing to the repository is https://github.com/WUSTL-CSPL/SMACK/commits/895f19b35350c5aded3362508c4a770f5e36342f.

A.2.3 Hardware dependencies

The attack framework can run on a machine with a moderate CPU, at least 16GB of available RAM, and a GPU with at least 8GB VRAM. The system was tested stable with AMD Ryzen 9 3900X 12-Core Processor accompanying RTX 3070Ti and 32GB memory. No other specific hardware is required, but the variance in hardware can lead to differences in run-time.

A.2.4 Software dependencies

SMACK was implemented in Python, and the environment was set up using Miniconda 4.12.0 on Ubuntu 22.0.4. The used machine learning framework is Pytorch, and other associated dependencies are encapsulated in a YAML file. For the installation process please see Section A.3. Due to the file size limit of GitHub, some of the files used for testing need to be downloaded from the Google Drive link, https://drive.google.com/file/d/12vUxRaIRDaD_prg8F-vpb5oUvWMOPqsl/view?usp=sharing, containing custom scripts and pre-trained models. Please refer to README.md for detailed guidance.

A.2.5 Benchmarks

The data required by the experiments are the generative model and speech audio used in adversarial optimization. For speech recognition, we provide two ASR cloud services, iFlytek and Google Speech-to-Text. For speaker recognition, we provide two state-of-the-art models (GMM-UBM and ivector-PLDA) as targets. The SR models are provided in the artifact, which can be found in the *./FAKEBOB* directory.

A.3 Set-up

A.3.1 Installation

Conda or Miniconda is recommended for setting up the environment. It can be installed via the official link https://docs.conda.io/en/latest/miniconda.html and the process can differ based on the user's OS. The commands for setting up the environment with the *smack.yml* file are:

```
$ cd <the_path_to_the_folder>
$ conda env create -f smack.yml
$ conda activate smack
```

For the setup of speaker recognition systems, we follow the existing work *FAKEBOB* and use the Kaldi toolkit. Notably, this process can be time-consuming and requires modification of the shell configuration file. Therefore, we wrote a dedicated tutorial detailing all steps in the *setup_SR.md* file.

A.3.2 Basic Test

The SMACK attack framework consists of two main components, the adapted genetic algorithm (AGA) and gradient estimation scheme (ES), each can be individually tested. Due to limited space, some printed outputs are omitted in this appendix and some of the commands are broken into multiple lines to fit the template. The full basic test can be found in the *README.md* document in the *Basic Tests* section, please copy the commands from there.

The command for basic tests of the AGA is:

```
$ python3 genetic.py
```

You are expected to see printed outputs comprising the unique ID of individuals in the population and their fitness value breakdown, including fitness value, confidence score, adversarial term value, and audio quality term value.

Similarly, the basic functionality of the gradient estimation part can be tested with:

```
$ python3 gradient.py
```

Note that the above two tests are based on speaker recognition systems, so they examine the setup and functionality of both attack algorithms and SR models. If the SR models are not properly setup prior to tests, the basic tests would fail automatically.

Besides, the normal functionality of the target models can be tested by running corresponding scripts. For ASR, we can use an adversarial example to test *iflytekASR* model:

```
$ python3 iflytek_ASR.py \
"SMACK_Examples/iflytekASR_THEY DID NOT HAVE \
A LIGHT.wav"
```

And you are expected to see the output as follows:

```
iflytek ASR Result after 1 connection retries:
THEY DID NOT HAVE A LIGHT
```

Similarly, the command and expected output for Google speech-to-text model is:

```
$ python3 google_ASR.py \
"SMACK_Examples/success_gmmSV_librispeech_p1089.wav"

Google ASR Result after 0 retries:
MY VOICE IS THE PASSWORD
```

For SR models, the testing command with the provided adversarial example is:

```
$ python3 speaker_sv.py \
"SMACK_Examples/success_gmmSV_librispeech_p1089.wav" \
gmmSV librispeech_p1089
```

And you are expected to see the output saying the tests on GMM-UBM models passed, with additional information on the decision, acceptance threshold, and score. The complete commands and associated outputs can be found in the "*README.md*" in the artifact.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): SMACK can be used to generate natural speech that misleads the state-of-the-art automatic speech recognition models, including commercial products such as iFlyTek and Google speech-to-text. This is proven by the experiment (E1) and (E2) described in Section 8.1 whose results are reported in Table 1 and Table 2.

(C2): SMACK can be used to generate natural speech that misleads the state-of-the-art speaker recognition models. The attack can also be achieved for the challenging inter-gender attack scenario, where the speaker of the adversarial example and targets are of different genders. This is proven by the experiment (E3) and (E4) described in Section 8.2 whose results are reported in Table 3.

A.4.2 Experiments

(E1): *[iFlyTek ASR Attack] [10 human-minutes + 4 compute-hour + 15GB disk]:*

Preparation: The environment installation is detailed in the previous section. Since the target models are commercial products, the speech recognition query is limited and comes with a cost. Please don't hesitate to contact us if you find the service no longer available. We will replace a valid token for your use.

Execution: In the attack against ASR systems, we provide two real-world speech recognition services as the target models, iFlytek and Google. In this experiment against iFlyTek, please use the following command:

```
$ python3 attack.py \
--audio "./Original_TheyDenyTheyLied.wav" \
--model iflytekASR \
--content "They deny they lied" \
--target "They did not have a light"
```

Results: You are expected the see printed outputs similar to the basic tests. All the intermediate audio files generated throughout the attack process are stored in the "*./SampleDir*" directory. An example terminal output produced by this attack is recorded in the "*SMACK_Examples/iflytekASR_THEY DID NOT HAVE A LIGHT.txt*" file, and the resulted adversarial example is provided as "*SMACK_Examples/iflytekASR_THEY DID NOT HAVE A LIGHT.wav*". The example is named as its transcription by the target model. To validate the adversarial example, please use the same command in the basic test:

```
$ python3 iflytek_ASR.py \
"SMACK_Examples/iflytekASR_THEY DID NOT \
HAVE A LIGHT.wav"
```

(E2): *[Google ASR Attack] [10 human-minutes + 1 compute-hour + 15GB disk]:*

Preparation: The environment setup is detailed in the previous section. The Google speech-to-text is a commercial cloud service that requires user token. It is already provided in the "*google_token.json*" file in the artifact and no other setup is needed as long as the token is still valid. Please contact us for a renewed token if it expires.

Execution: Similar to the attack against iFlyTek, please use the following command to launch the attack against Google ASR:

```
$ python3 attack.py \
--audio "./Original_SamiGotAngry.wav" \
--model googleASR \
--content "Sami got angry" \
--target "Send me that"
```

Results: The expected results are similar to the experiment (E1). We also provide a sample adversarial example "*SMACK_Examples/googleASR_SEND ME THAT.wav*", along with its terminal prints recorded in the "*SMACK_Examples/googleASR_SEND ME THAT.txt*" file. To validate the adversarial example, please use the command:

```
$ python3 google_ASR.py \
"SMACK_Examples/googleASR_SEND ME THAT.wav"
```

(E3): *[GMM-based SR Attack] [10 human-minutes + 8 compute-hour + 15GB disk]:*

Preparation: It requires the setup of both attack framework and speaker verification encapsulated in the Kaldi toolkit. The detailed guidance for installing Kaldi and associated dependencies are included in the "*setup_speakerRecog.md*" file in the artifact.

Execution: In this attack, we target a well-established model GMM-UBM deployed with the Kaldi toolkit. The command for running the attack is as follows:

```
$ python3 attack.py \
--audio "./Original_MyVoiceIsThePassword.wav" \
--model gmmSV
--content "My voice is the password" \
--target librispeech_p1089
```

By using this command, we conduct an inter-gender attack, that is, the reference audio "*./Original_MyVoiceIsThePassword.wav*" is uttered by a woman while the target speaker *librispeech_p1089* is a man.

Results: All the intermediate audio files generated throughout the attack process are stored in the "*./SampleDir*" directory. The adversarial examples that successfully achieve the adversarial goal will be stored in the "*./SuccessDir*" directory. The terminal prints also reveal the process of that attack and intermediate results (such as loss values). An adversarial example generated by the attack is provided in "*SMACK_Examples/success_gmmSV_librispeech_p1089.wav*, and its success can be validated in the basic test 4. The associated printed output throughout the optimization process is also provided in "*SMACK_Examples/success_gmmSV_librispeech_p1089.txt*.

(E4): *[ivector-based SR Attack] [10 human-minutes + 8 compute-hour + 15GB disk]:*

Preparation: The setup required by this experiment is the same with experiment (E3).

Execution: In this attack, we target another well-established model ivector-PLDA deployed with the Kaldi toolkit. The command for running the attack is as follows:

```
$ python3 attack.py \
--audio "./Original_MyVoiceIsThePassword.wav" \
--model ivectorCSI
--content "My voice is the password" \
--target librispeech_p1089
```

The launched attack specified by this command is also an inter-gender attack.

Results: The expected results are similar to that of the experiment (E3). We also provide two adversarial examples generated by the attack in "*SMACK_Examples/success_ivectorCSI_librispeech_p1089.wav*", "*SMACK_Examples/success_ivectorCSI_librispeech_p1089_1.wav*", and their success can be validated in the basic test. The associated printed output throughout the optimization process is also provided in "*SMACK_Examples/success_ivectorCSI_librispeech_p1089.txt*.

A.5 Version

Based on the LaTeX template for Artifact Evaluation V20220926. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/usenixsec2023/.