Augment cybersecurity through A.I.

From a software engineer perspective

Mihail-Iulian Pleșa
Special Telecommunications Service

Content

- Detect obfuscated JavaScript files
- Malware detection
- Malware generation
- Twitter phishing
- Voice impersonation
- DeepExploit

Detect obfuscated JavaScript

Dataset

- Dataset
  - 1477 obfuscated files
  - 1898 not obfuscated files

Detect obfuscated JavaScript

Model

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 120, 16)           989968

 bidirectional (Bidirectiona  (None, 64)               12544
 l)

 dense (Dense)               (None, 32)                2080

 dense_1 (Dense)             (None, 1)                 33

=================================================================
Total params: 1,004,625
Trainable params: 1,004,625
Non-trainable params: 0
```

Detect obfuscated JavaScript

Solution

- https://colab.research.google.com/drive/1qcA4rt1LawqCt59bGnKJiSLOMBVEVByX?usp=sharing

Malware detection

Problem

- Dataset
  - 5560 samples of malware android apps
  - 9476 samples of benign android apps
- 215 features for each sample

Malware detection

Feature examples

transact	API call signature
onServiceConnected	API call signature
bindService	API call signature
attachInterface	API call signature
ServiceConnection	API call signature
android.os.Binder	API call signature
SEND_SMS	Manifest Permission
Ljava.lang.Class.getCanonicalName	API call signature
Ljava.lang.Class.getMethods	API call signature
Ljava.lang.Class.cast	API call signature
Ljava.net.URLDecoder	API call signature
android.content.pm.Signature	API call signature
android.telephony.SmsManager	API call signature
READ_PHONE_STATE	Manifest Permission
getBinder	API call signature
ClassLoader	API call signature

Malware detection

Model

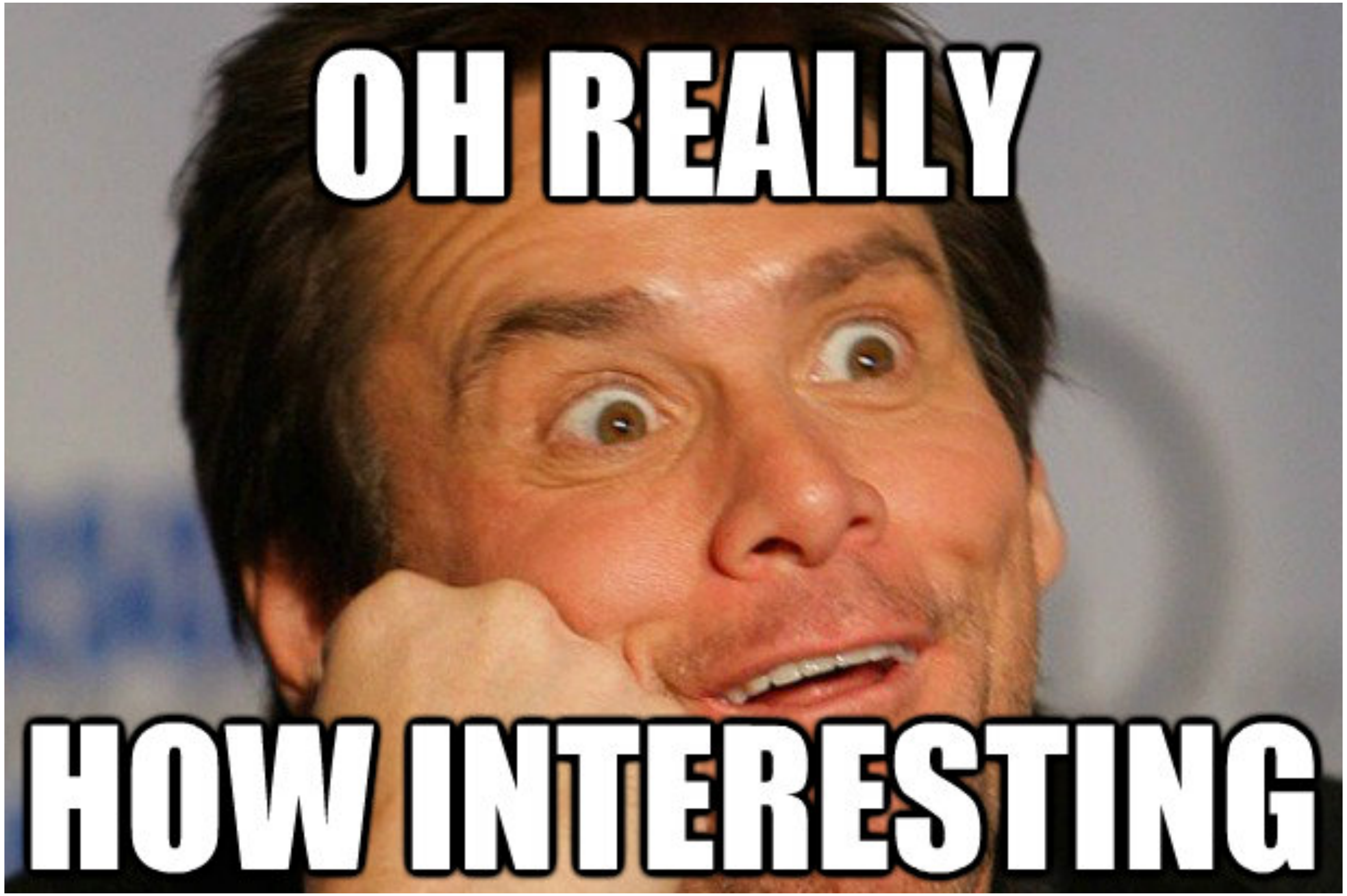
```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d (Conv1D)             (None, 211, 32)           192

 max_pooling1d (MaxPooling1D  (None, 105, 32)          0
 )

 conv1d_1 (Conv1D)           (None, 101, 32)           5152

 max_pooling1d_1 (MaxPooling  (None, 50, 32)           0
 1D)

 conv1d_2 (Conv1D)           (None, 46, 64)            10304

 global_max_pooling1d (Globa  (None, 64)               0
 lMaxPooling1D)

 dense (Dense)               (None, 256)               16640

 dense_1 (Dense)             (None, 1)                 257

=================================================================
Total params: 32,545
Trainable params: 32,545
Non-trainable params: 0
_________________________________________________________________
```

Malware detection

Solution

- https://colab.research.google.com/drive/1Yzcm5AFB-0GNd0mje07frAlXxk7uHtEc?usp=sharing

Features?



Malware detection

MalConv

Malware Detection by Eating a Whole EXE

Edward Raff[1,3,4], Jon Barker[2], Jared Sylvester[1,3], Robert Brandon[1,3,4]
Bryan Catanzaro[2], Charles Nicholas[4]
[1]Laboratory for Physical Sciences, [2]NVIDIA, [3]Booz Allen Hamilton, [4]University of Maryland, Baltimore County
{edraff,jared,rbrandon}@lps.umd.edu, {jbarker,bcatanzaro}@nvidia.com, nicholas@umbc.edu

Abstract

In this work we introduce malware detection from raw byte sequences as a fruitful research area to the larger machine learning community. Building a neural network for such a problem presents a number of interesting challenges that have not occurred in tasks such as image processing or NLP. In particular, we note that detection from raw bytes presents a sequence problem with over two million time steps and a problem where batch normalization appear to hinder the learning process. We present our initial work in building a solution to tackle this problem, which has linear complexity dependence on the sequence length, and allows for interpretable sub-regions of the binary to be identified. In doing so we will discuss the many challenges in building a neural network to process data at this scale, and the methods we used to work around them.

inside a specially instrumented environment, such as a customized Virtual Machine (VM), which introduces high computational requirements. Furthermore, in some cases it is possible for malware to detect when it is being analyzed. When the malware detects an attempt to analyze it, the malware can alter its behavior, allowing it to avoid discovery (Raffetseder, Kruegel, and Kirda 2007; Garfinkel et al. 2007; Carpenter, Liston, and Skoudis 2007). Even when malware does not exhibit this behavior, the analysis environment may not reflect the target environment of the malware, creating a discrepancy between the training data collected and real life environments (Rossow et al. 2012). While a dynamic analysis component is likely to be an important component for a long term solution, we avoid it at this time due to its added complexity.

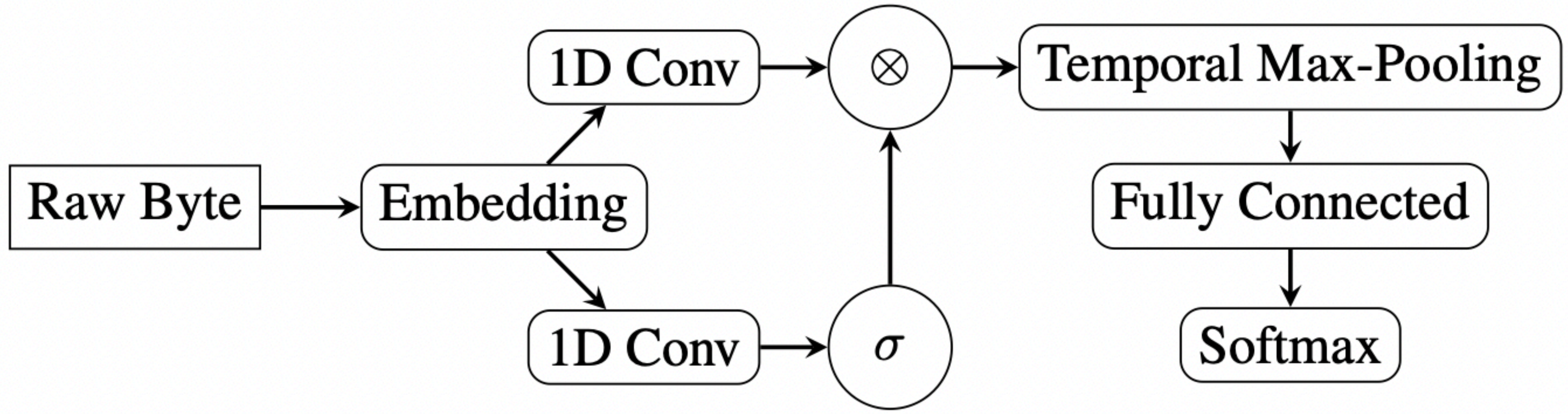
Malware Detection

MalConv - Dataset

- Dataset
  - Group B (provided by industry partner)
    - 200000 benign files
    - 200000 malware files
  - Group A
    - 21854 benign files (Clean Windows Install)
    - 43967 malware files (VirusShare)

Malware Detection

MalConv - Model



Malware Detection

MalConv - Idea

- MS-DOS Header: Constant position
- PE Header: Variable position
- Functions can be rearranged in any order
- The meaning of a byte is generated by the context (embeddings)

Malware Detection

MalConv - Solution

- Already trained
- https://github.com/j40903272/MalConv-keras

Generating Malware

MalGAN

Generating Adversarial Malware Examples for Black-Box Attacks Based on GAN

Weiwei Hu and **Ying Tan***
Key Laboratory of Machine Perception (MOE), and Department of Machine Intelligence
School of Electronics Engineering and Computer Science, Peking University, Beijing, 100871 China
{weiwei.hu, ytan}@pku.edu.cn

Abstract

Machine learning has been used to detect new malware in recent years, while malware authors have strong motivation to attack such algorithms.Malware authors usually have no access to the detailed structures and parameters of the machine learning models used by malware detection systems, and therefore they can only perform black-box attacks. This paper proposes a generative adversarial network (GAN) based algorithm named MalGAN to generate adversarial malware examples, which are able to bypass black-box machine learning based detection models. MalGAN uses a substitute detector to fit the black-box malware detection system. A generative network is trained to minimize the generated adversarial examples' malicious probabilities predicted by the substitute detector. The superiority of MalGAN over traditional gradient based adversarial example generation algorithms is that MalGAN is able to decrease the detection rate to nearly zero and make the retraining based defensive method against adversarial examples hard to work.

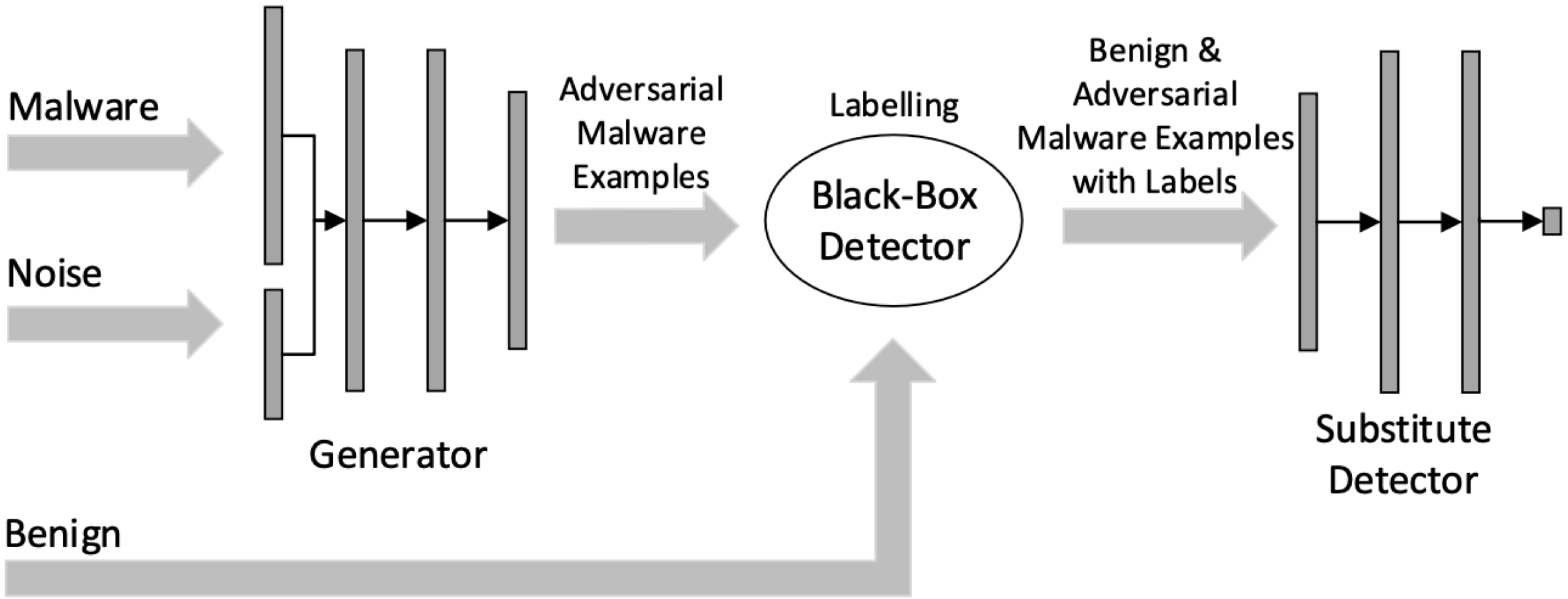
Many machine learning algorithms are very vulnerable to intentional attacks. Machine learning based malware detection algorithms cannot be used in real-world applications if they are easily to be bypassed by some adversarial techniques.

Recently, adversarial examples of deep learning models have attracted the attention of many researchers. Szegedy et al. added imperceptible perturbations to images to maximize a trained neural network's classification errors, making the network unable to classify the images correctly [Szegedy *et al.*, 2013]. The examples after adding perturbations are called adversarial examples. Goodfellow et al. proposed a gradient based algorithm to generate adversarial examples [Goodfellow *et al.*, 2014b]. Papernot et al. used the Jacobian matrix to determine which features to modify when generating adversarial examples [Papernot *et al.*, 2016c]. The Jacobian matrix based approach is also a kind of gradient based algorithm.

Grosse et al. proposed to use the gradient based approach to generate adversarial Android malware examples [Grosse *et al.*, 2016]. The adversarial examples are used to fool a neural network based malware detection model. They assumed that attackers have full access to the parameters of the malware detection model. For different sizes of neural networks, the misclassification rates after adversarial crafting range from 40% to 84%.

Generating Malware

MalGAN



Generating Malware

MalGAN

- Black Box Detector:
    - LR
    - SVM
    - DT
    - MLP (MalConv)
    - ...

Generating Malware

MalGAN - Solution

- https://github.com/yanminglai/Malware-GAN

Twitter Phishing

Dataset

- Dataset
  - 1559 Elon Musk Tweets

Twitter Phishing

Model

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 53, 100)           432400

 bidirectional (Bidirectiona  (None, 600)              962400
 l)

 dense (Dense)               (None, 4324)              2598724

=================================================================
Total params: 3,993,524
Trainable params: 3,993,524
Non-trainable params: 0
_________________________________________________________________
```

Twitter Phishing

Solution

- https://colab.research.google.com/drive/14VRSahmz1g763_gIfunEfutpKcljkUih?usp=sharing

Twitter Phishing

GPT-2

- https://github.com/borisdayma/huggingtweets

Voice impersonation

Transfer Learning from Speaker Verification to Multispeaker Text-To-Speech Synthesis

Ye Jia* **Yu Zhang*** **Ron J. Weiss*** **Quan Wang** **Jonathan Shen** **Fei Ren**
Zhifeng Chen **Patrick Nguyen** **Ruoming Pang** **Ignacio Lopez Moreno** **Yonghui Wu**
Google Inc.
{jiaye,ngyuzh,ronw}@google.com

Abstract

We describe a neural network-based system for text-to-speech (TTS) synthesis that is able to generate speech audio in the voice of different speakers, including those unseen during training. Our system consists of three independently trained components: (1) a *speaker encoder network*, trained on a speaker verification task using an independent dataset of noisy speech without transcripts from thousands of speakers, to generate a fixed-dimensional embedding vector from only seconds of reference speech from a target speaker; (2) a sequence-to-sequence *synthesis network* based on Tacotron 2 that generates a mel spectrogram from text, conditioned on the speaker embedding; (3) an auto-regressive WaveNet-based *vocoder network* that converts the mel spectrogram into time domain waveform samples. We demonstrate that the proposed model is able to transfer the knowledge of speaker variability learned by the discriminatively-trained speaker encoder to the multispeaker TTS task, and is able to synthesize natural speech from speakers unseen during training. We quantify the importance of training the speaker encoder on a large and diverse speaker set in order to obtain the best generalization performance. Finally, we show that randomly sampled speaker embeddings can be used to synthesize speech in the voice of novel speakers dissimilar from those used in training, indicating that the model has learned a high quality speaker representation.

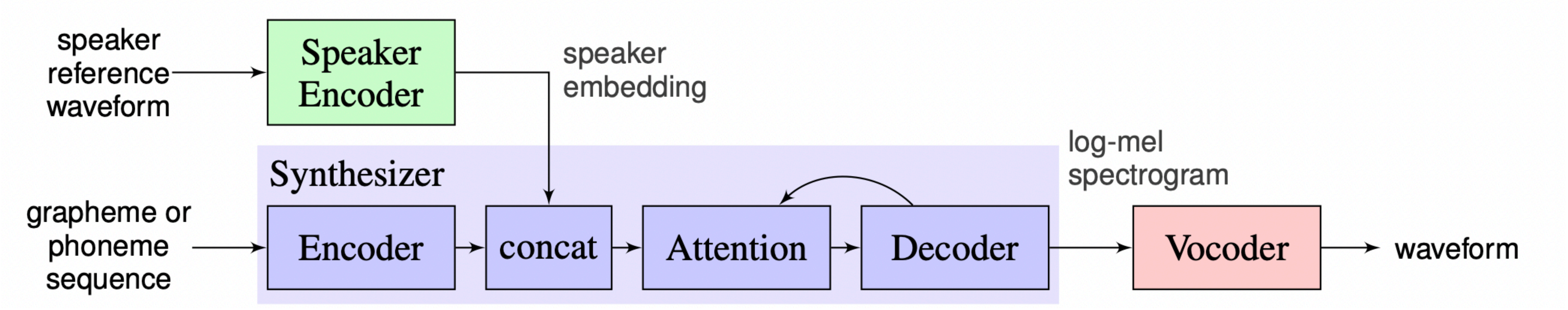
Voice impersonation

Dataset

- VCTK
  - 44 hours from 109 speakers
- LibriSpeech
  - 436 hours from 1172 speakers

Voice impersonation

Model



Voice impersonation

Solution

- https://github.com/CorentinJ/Real-Time-Voice-Cloning

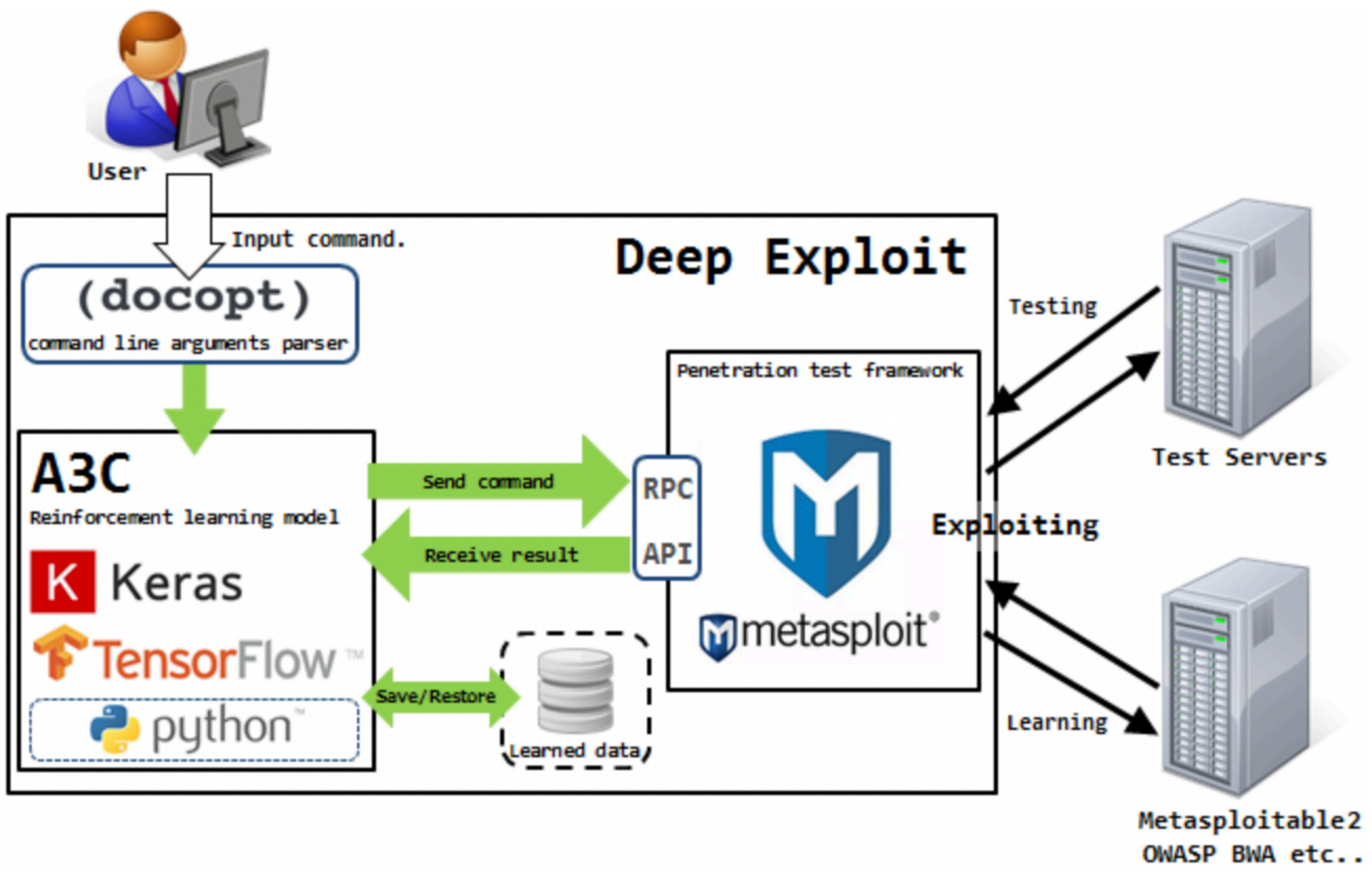
Penetration testing

DeepExploit

- Reinforcement learning
- Improve efficiency
- Continuous learning

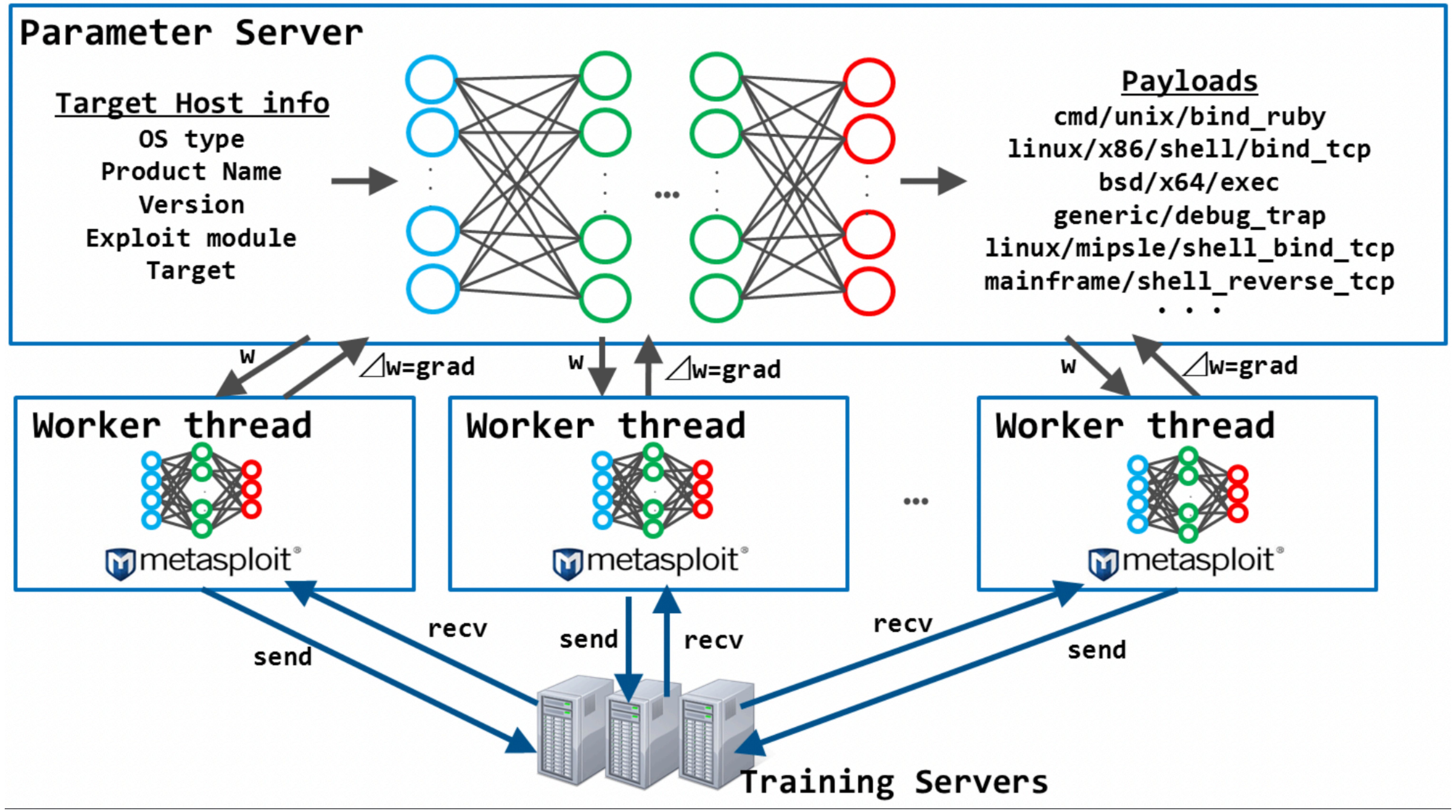
Penetration testing

DeepExploit - Architecture



Penetration testing

DeepExploit - Learning

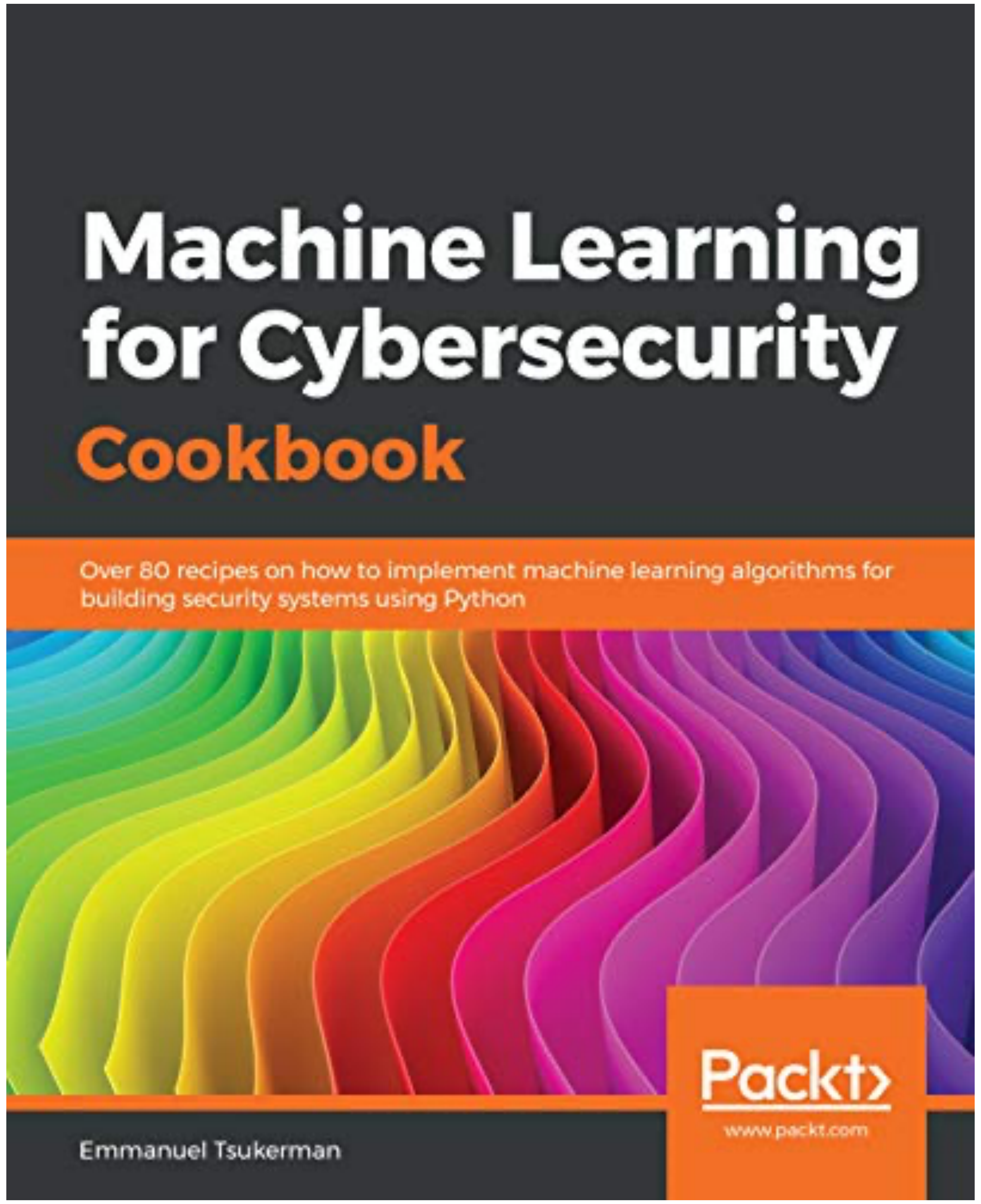


Penetration testing

DeepExploit - Solution

- https://github.com/13o-bbr-bbq/machine_learning_security/tree/master/DeepExploit

Resources


Machine Learning for Cybersecurity
Cookbook
Over 80 recipes on how to implement machine learning algorithms for building security systems using Python
Packt>
www.packt.com
Emmanuel Tsukerman

Resources

- https://github.com/PacktPublishing/Machine-Learning-for-Cybersecurity-Cookbook
- https://www.kaggle.com/datasets
- https://laurencemoroney.com

Conclusions

- The lack of data
- Large models are needed sometimes
- Augument not automate
- The security mindset

Thank you!