Summer 8-2020

Secure Mobile Computing by Using Convolutional and Capsule Deep Neural Networks

Rui Ning
Old Dominion University, rui.ning.1990@gmail.com

SECURE MOBILE COMPUTING BY USING CONVOLUTIONAL AND CAPSULE DEEP NEURAL NETWORKS

by

Rui Ning
B.Sc. May 2011, Lanzhou University
M.Sc. May 2016, University of Louisiana at Lafayette

A Dissertation Submitted to the Faculty of
Old Dominion University in Partial Fulfillment of the
Requirements for the Degree of

DOCTOR OF PHILOSOPHY

ELECTRICAL & COMPUTER ENGINEERING

OLD DOMINION UNIVERSITY
August 2020

Approved by:

Hongyi Wu (Director)

Chunsheng Xin (Member)

Cong Wang (Member)

Jiang Li (Member)

ABSTRACT

SECURE MOBILE COMPUTING BY USING CONVOLUTIONAL AND CAPSULE DEEP NEURAL NETWORKS

Rui Ning
Old Dominion University, 2020
Director: Dr. Hongyi Wu

Mobile devices are becoming smarter to satisfy modern user's increasing needs better, which is achieved by equipping divers of sensors and integrating the most cutting-edge Deep Learning (DL) techniques. As a sophisticated system, it is often vulnerable to multiple attacks (side-channel attacks, neural backdoor, etc.). This dissertation proposes solutions to maintain the cyber-hygiene of the DL-Based smartphone system by exploring possible vulnerabilities and developing countermeasures.

First, I actively explore possible vulnerabilities on the DL-Based smartphone system to develop proactive defence mechanisms. I discover a new side-channel attack on smartphones using the unrestricted magnetic sensor data. I demonstrate that attackers can effectively infer the Apps being used on a smartphone with an accuracy of over 80%, through training a deep Convolutional Neural Networks (CNN). Various signal processing strategies have been studied for feature extractions, including a tempogram based scheme. Moreover, by further exploiting the unrestricted motion sensor to cluster magnetometer data, the sniffing accuracy can increase to as high as 98%. To mitigate such attacks, I propose a noise injection scheme that can effectively reduce the App sniffing accuracy to only 15% and, at the same time, has a negligible effect on benign Apps.

On the other hand, I leverage the DL technique to build reactive malware detection schemes. I propose an innovative approach, named CapJack, to detect in-browser malicious cryptocurrency mining activities by using the latest CapsNet technology. To the best of our knowledge, this is the first work to introduce CapsNet to the field of malware detection

through system-behavioural analysis. It is particularly useful to detect malicious miners under multitasking environments where multiple applications run simultaneously.

Finally, as DL itself is vulnerable to model-based attacks, I proactively explore possible attacks against the DL model. To this end, I discover a new clean label attack, named Invisible Poison, which stealthily and aggressively plants a backdoor in neural networks (NN). It converts a trigger to noise concealed inside regular images for training NN, to plant a backdoor that can be later activated by the trigger. The attack has the following distinct properties. First, it is a black-box attack, requiring zero-knowledge about the target NN model. Second, it employs "invisible poison" to achieve stealthiness where the trigger is disguised as "noise" that is therefore invisible to human, but at the same time, still remains significant in the feature space and thus is highly effective to poison training data.

ACKNOWLEDGEMENTS

I want to express my gratitude to the Electrical and Computer Engineering Department and the Old Dominion University Research Foundation for financial support during my Ph.D. study.

I would like to pay my special regards to my advisor, Dr. Hongyi Wu, for advising me with great understanding and patience throughout my entire Ph.D journey. I sincerely thank Dr. Wu, not only for his guidance of my research, but also for his role model of being a humble, kind, and hardworking person. Without his endless support and help, I would never accomplish this work.

I would like to express my sincere appreciation to my committee members Dr. Chunsheng Xin, Dr. Cong Wang, and Dr. Jiang Li, for their valuable time and guidance. You inspired me to explore exciting research ideas and guided me to keep myself on the right path.

I want to acknowledge my labmates for their excellent collaboration. You supported me greatly and were always willing to help me whenever I needed it.

In addition, I would like to thank the support and great love of my parent, wife, and sister. You are always there for me. Besides, I deeply appreciate my daughter for providing a happy distraction to rest my mind outside of my research.

TABLE OF CONTENTS

Page

Chapter Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

CHAPTER 1

INTRODUCTION

Mobile devices have become constant companions in our daily lives. People are not just using their mobile devices at work and home – they are living on them. People rely on smartphone applications (Apps) for communication, social networking, entertainment, banking, shopping, navigation, healthcare, and more. For many people, every day begins and ends with checking their smartphones. As more and more personal data are stored on and processed and transmitted by smartphones, they are becoming an increasingly attractive target for cybercriminals.

The modern mobile devices are becoming "smarter" to satisfy the increasing need of users. Machine Learning techniques, Deep Learning (DL) specifically, have been adopted for productivity and smarter decision making. They usually cooperate with a range of integrated sensors, such as accelerometer, gyroscope, magnetometer, GPS, gravity sensor, barometer, microphone, ambient light sensor, and proximity sensor, to extract the knowledge of the current state. As all are resulting in a sophisticated system, it is often vulnerable to multiple attacks, including side-channel attacks (malicious use of on-device sensors), neural backdoor, and data poisoning attacks, etc.

1.1 ATTACKS AGAINST DL-BASED SMARTPHONE SYSTEM

Side-channel Attack A range of work has attempted to construct side-channel attacks to infer app usage on smartphones. Various methods have been leveraged, for instance, power usage [1], system behaviour [2], and network traffic [3, 4]. In [1, 2], the authors developed malicious Android Apps to collect system information, such as current, voltage, network state, CPU and memory usage, from a victim. The collected data were analyzed,

and machine learning techniques were used to identify the Apps on the victim's device. In [3,4], the authors designed learning systems to automatically fingerprint an App using the encrypted network traffic (e.g., IP, protocol type, length, etc.). However, their approaches require to either maliciously deploy an additional App [1, 2] or a traffic sniff device [3, 4] to collect data on the victim's device. It is also worth to mention that their approaches perform poorly in scenarios that the Apps have minimal network traffic, or have random system behaviour caused by multithreading.

In addition, Recent studies introduced several attacks due to the malicious use of magnetometer and motion sensors. For instance, a range of work [1, 5, 6] raised the idea of cyber-vulnerability in 3D printing, where a piece of malicious software can infer the design files by sniffing magnetometer. Furthermore, two recent works [7, 8] proposed utilizing motion sensors in smart devices to infer a user's typed words or passwords. Similarly, Narain et al. [9, 10] demonstrated that the accelerometer and the gyroscope could be used to infer car driving routes and fingerprint devices.

Compared with traditional side-channel attacks, which usually require professional tools to sample side-channel signals, the mobile side-channel signal sniffing can be conducted easily by leveraging APIs provided by the operating system of mobile devices. In addition, since the collected signal data can be transferred to the attacker in real-time through the Internet, the adversary can perform the attack world-widely as long as they are online. However, the challenging part of the mobile side-channel attack is how to efficiently extract useful knowledge from the multi-modal signals from sensors with limited accuracy. To this end, attackers reach out to find a more powerful, efficient, and adaptive tool for feature extraction.

DL-based Model Attack While deep learning is embraced as an important tool for efficiency and productivity, it is becoming an increasingly attractive target for cybercriminals. For example, adversarial example (AE) [11–14] is one of the most widely studied attacks. It adds small perturbation to a clean image to fool a target *neural network* (NN) to misclassify

an image.

This attack usually requires full knowledge of the target NN model or abundant similarly-distributed training data to train a substitute NN. Moreover, an AE is narrowly targeted at individual input samples. Previous studies [15, 16] have shown that a small change in AE could render it ineffective. For instance, if an AE is interfered with by physical noises introduced by a display, camera, or viewing angle change, it would fail to trigger the targeted misclassification [15, 17].

Studies have shown a class of more robust attacks by planting a backdoor in NN models [18, 19]. The basic idea is to create a unique pattern and include it in training data. The trained NN thus contains a backdoor. Whenever the trigger appears in the input image, it activates the backdoor to misclassify the input to a category targeted by the attacker. Compared with AE, the backdoor attack is more robust, especially under physical noises. However, to plant a backdoor, the targeted model must be trained by the attacker using manipulated training data and then offered to targeted users. This constraint makes the attack less realistic.

The quest continues with the discovery of clean label attack [20, 21], which still targets at creating a backdoor, but does not make the strong assumption that the targeted model must be trained by the attacker. Instead, it aims to poison all or part of the training data used by the victim in order to plant a backdoor. The key to success in such an attack is to be stealthy, such that the victim cannot detect that the training data have been poisoned. To this end, the training data can be constructed by using AE samples.

However, it inherits the limitations of AE as it requires either a white-box [20, 22] or a substitute NN [21, 23]. Another approach is to combine AE with a watermarked "trigger" [20] to poison the images [23] to strive for a higher transferability over different NNs. In these approaches, the triggers are, however, visible to human, and thus are likely to be discovered by alerted users for image collection. Moreover, experiments show that it degrades the performance of the targeted model.

1.2 DISSERTATION CONTRIBUTIONS

In this dissertation, I have developed and demonstrated a systematical methodology of discovering possible vulnerabilities and developing countermeasures on the DL-based smartphone system. Though it is not applicable for me to exhaust all the weaknesses and build corresponding defence mechanisms. The methodology of the demonstrated examples in my dissertation can still guide and motivate future researches in this field.

1. **Proactively discovering side-channel vulnerabilities.**

   It is crucial to discover new side-channel vulnerabilities before it causes damages. To my best knowledge, my work is the first that discovers and reports the correlation between magnetometer readings and LED displays on smartphones. Based on this observation, I demonstrate a newfound side-channel attack, where the attacker can sniff mobile Apps through magnetometer data. In particular, I devise deep Convolutional Neural Networks (CNN) that can be trained to effectively infer the Apps installed on the smartphone and their usage information. I implement the attack on iPhone 7 Plus and Samsung Galaxy 7 and carry out extensive experiments, showing that the attack can achieve an accuracy of 80-90% based on the magnetometer data.

   Furthermore, I discover that the orientation data is closely correlated with the magnetometer readings. To this end, I use the orientation of the smartphone as an alternative to train and test the CNN models. The performance is only slightly lower than the original approach based on magnetometer data. This finding enables even more pervasive attacks since the orientation data can be readily obtained by integrating a 4-line Javascript code in attacker's websites, and the Javascript can continuously acquire the orientation data even in the background. Besides showing this newfound side-channel attack, I also discuss viable methods to mitigate it by injecting a minimal amount of noise into the magnetometer or orientation data.

2. **Using deep learning technology to build side-channel defense mechanisms.**

   Deep learning has proven its success in computer vision, but it not trivial to directly adopt it to our field. This is because the sensor data is spatial and contains rich time-related information, while the digital image is continuous and time irrelevant. Therefore, I am exploring possible methods to adopt cutting-edge deep learning techniques into the side-channel attack field seamlessly.

   I propose an innovative approach to detect malicious cryptocurrency mining activities by using the latest CapsNet technology. To the best of my knowledge, this is the first work to introduce CapsNet to the field of malware detection. It is particularly effective to detect malicious miners under multitasking environments. Built upon the success of the CapsNet-based approach, I further develop a two-layer classification system, named *CapJack*, which can effectively transform a pre-trained model to detect miners on new devices. This is intrinsically important to achieve practical usability, given the wide variety of devices used by victims. The work delivers a well-engineered prototype. The experiments reveal valuable empirical insights into the design space for miner detection and the application of CapsNet for detecting malicious mining activities. Experimental data show the appealing performance of CapJack, with a detection rate of as high as 87% instantly and 99% within a window of 11 seconds.

3. **Exploring vulnerabilities of deep learning techniques.** As the DL technology plays as an essential role in my previous works, its robustness becomes crucial. Therefore, I reach out to explore the vulnerabilities of deep neural networks. I discover a new clean label attack, named *Invisible Poison*, which stealthily and aggressively plants a backdoor in neural networks (NN). It converts a trigger to noise concealed inside regular images for training NN, to plant a backdoor that can be later activated by the trigger.

1.3 OUTLINE OF THE DISSERTATION

This dissertation is organized into the following chapters:

- Chapter 1: Introduction to the problem, existing works, and our approach's contributions.
- Chapter 2: We discuss a new found side-channel attack against smartphones and its countermeasures.
- Chapter 3: We discuss the methodology of introducing CapsNet to the field of malware detection.
- Chapter 4: We shortly present our current work on discovering vulnerabilities of deep neural networks and future works.
- Chapter 5: Conclusion.

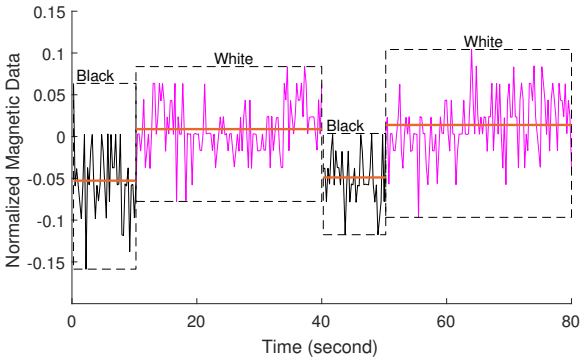
CHAPTER 2

DISCOVERING MOBILE SIDE-CHANNEL VULNERABILITIES: SNIFFING MOBILE APPS IN MAGNETIC FIELD THROUGH DEEP CONVOLUTIONAL NEURAL NETWORKS (DEEPMAG)

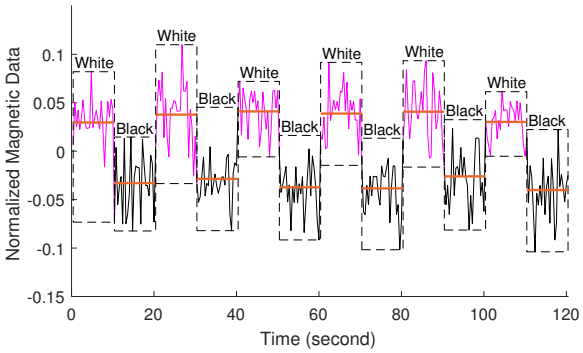
This chapter reports a new side-channel attack on smartphones using the unrestricted magnetic sensor data. We demonstrate that attackers can effectively infer the Apps being used on a smartphone with an accuracy of over 80%, through training a deep Convolutional Neural Networks (CNN). Various signal processing strategies have been studied for feature extractions, including a tempogram based scheme. Moreover, by further exploiting the unrestricted motion sensor to cluster magnetometer data, the sniffing accuracy can increase to as high as 98%. To mitigate such attacks, we propose a noise injection scheme that can effectively reduce the App sniffing accuracy to only 15% and at the same time has a negligible effect on benign Apps.

2.1 PRELIMINARY EXPERIMENTS AND MOTIVATIONS

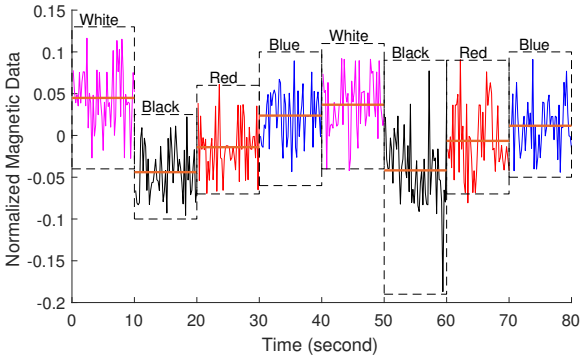
The quest begins with the observation of a subtle correlation between an LED display and its surrounding magnetic field. For example, in our first experiment, we display a black image on a 27 inch LED PC monitor for 20 seconds, followed by a white image for 60 seconds. We repeat the pattern for a number of rounds. In the meantime, an iPhone 7 Plus is placed in front of the monitor about $10cm$ away to measure the magnetic field. We observe a noticeable change in the magnetic field while switching between the two images (see Fig. 1(a)).



(a)



(b)



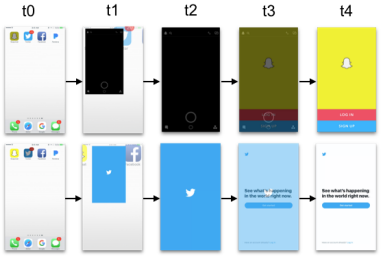
(c)

Figure 1. Magnetometer readings on Y-axis. (a) Change of magnetometer reading due to PC LED display, where the black and magenta waveforms correspond to the black and white images, respectively. (b) Change of magnetic field due to smartphone LED display (black and white). (c) Change of magnetic field while the smartphone sequentially displays white-black-red-blue. The solid orange line in each dashed rectangle box represents the mean of the signal within the box.

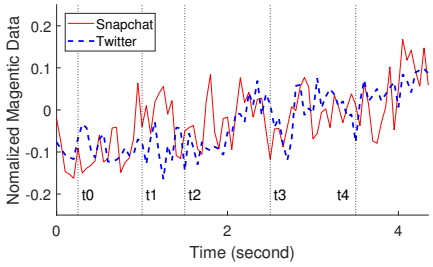
The above phenomenon motivates us to further explore how the LED display on a smartphone would affect its magnetometer readings. To this end, we carry out a similar experiment by displaying the black and white images on the iPhone 7 Plus while recording the data captured by the magnetometer on the same phone. Compared with the previous setting, we expect more stable results since the distance between the LED display and the magnetometer is shorter and their relative orientation is fixed. The experimental data are depicted in Fig. 1(b), demonstrating significant changes in magnetic field when different images are displayed on the phone.

Fig. 1(c) further shows the change of magnetometer readings while the smartphone displays four different colors, white-black-red-blue, in sequence. While it is beyond our scope to fully model this physical phenomenon, it is largely due to the fact that different bias voltages are used when LED displays different colors, which accordingly lead to the change of the magnetic field [24].

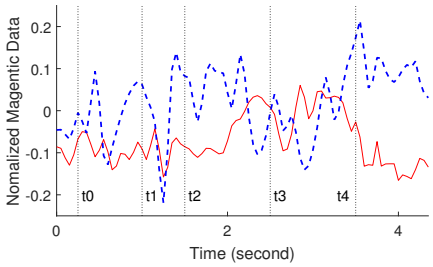
The results shown in Fig. 13 demonstrate the correlation between colors on LED display and surrounding magnetic field. Since different Apps often adopt different graphic designs that mix different color patterns, we speculate that they also induce different magnetometer readings. In particular, when one clicks on an App's icon, a unique welcome-page will be displayed till the App is fully open. The corresponding changes in magnetic field can be measured by the integrated magnetometer. Fig. 2 illustrates the averaged magnetometer readings over the period for opening two popular Apps, i.e., Snapchat and Twitter, on an iPhone 7 Plus. As can be seen, they differ dramatically on at least one axis (in this case, Y-axis). This is because Snapchat adopts predominantly a bright yellow color while Twitter uses blue. We have verified that the above observations are repeatable on different smartphones and models. More results will be presented in Secs. 2.2-2.4. These preliminary experimental data indicate that different Apps are likely associated with unique magnetic signatures. Therefore, if one can acquire magnetometer data, he can potentially infer the Apps running on a smartphone.



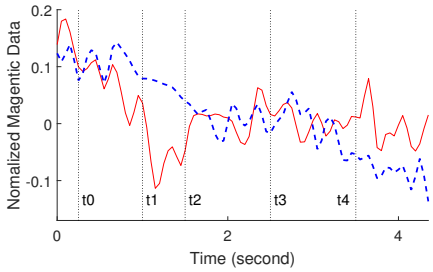
(a) Screen shots when opening Snapchat and Twitter.



(b) Magnetometer X-axis readings for Snapchat and Twitter.



(c) Magnetometer Y-axis readings for Snapchat and Twitter.



(d) Magnetometer Z-axis readings for Snapchat and Twitter.

Figure 2. Welcome pages and magnetometer readings of popular Apps.

2.2 MAGNETOMETER-BASED APP SNIFFING (MAS)

In this section, we present App sniffing solely based on the readings from magnetometer. The preliminary observations presented in Sec. 3.1 indicate that different Apps are likely

to induce different magnetic field. However, it remains a nontrivial problem to identify the unique magnetic signature for each App and accordingly infer the Apps according to magnetometer readings. The fundamental challenge lies in the facts that the magnetometer data exhibit noise and randomness and that the Apps' graphic designs often incorporate complex combination of color patterns, rendering simple classification methods infeasible. To this end, we propose to exploit the powerful deep *CNN* to classify magnetometer data and to infer the corresponding Apps.

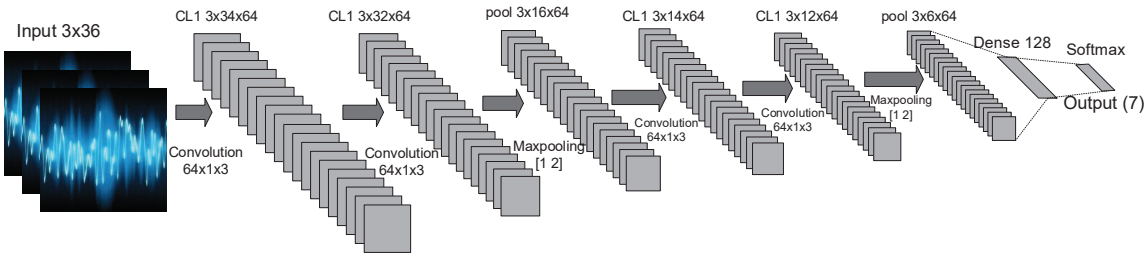


Figure 3. An example of the proposed deep CNN architectures.

2.2.1 DEEP CNN MODELS FOR MAGNETOMETER-BASED APP SNIFFING

Magnetometer data can be recorded when a user opens an App. In our preliminary exploration, we have considered the seven most commonly used Apps, i.e., Twitter, Snapchat, Pandora, Netflix, Google Maps, Chase Bank, and HBO. The recording process essentially collects a sequence of magnetometer samples during the interval from clicking on an App to the time when the welcome page of the App is fully displayed on the mobile screen. Fig. 2 shows the examples for opening Snapchat and Twitter, respectively. Different phones may have different sampling rates of their magnetometers. For instance, Samsung Galaxy S7 and iPhone 7 Plus sample their magnetometers at the rate of 20Hz and 100Hz, respectively. As to be shown later, the sampling rate has negligible impact on the effectiveness of App

sniffing.

Since the change of magnetic field is relatively small, we preprocess the raw magnetometer data by using a de-noising function (e.g., *wden* available in MATLAB [25]). It decomposes the signal into wavelets and performs thresholding on wavelets coefficients.

We set the function at the denoise-level 5 with soft thresholding rule for the universal threshold $\sqrt{2\ln(\cdot)}$. Then, we normalize the de-noised data by subtracting its mean and dividing it by its vector's norm. Figs. 2(b)-(d) illustrate the normalized magnetometer readings of Snapchat and Twitter on X, Y, and Z axis. While the figure only shows about 4 seconds, the total recording time is 8.25 seconds including some overhead before and after the App is opened.

Thus on an iPhone with a sampling rate of 100Hz, each magnetometer data would have a dimension of 825×3. Directly feeding such high dimensional data into the CNN yields poor results (less than 50% accuracy) because drastic changes over such a large span of 825 sample points may easily overwhelm a neural net's representational capability. To this end, we adopt a sliding window approach with a window size of W sample points to slide over the time series. Each pair of adjacent slices has an overlap of P sample points. For example, assume $W = 36$ and $P = 31$. If we have 100 original data (each with a dimension of 825×3), they will be converted to $15,800$ sliced data each with a dimension of 36×3. The sliced data are labeled with corresponding Apps for training and testing as to be discussed next.

We have explored several deep CNN architectures to sniff Apps based on magnetometer data. In contrast to computer vision (that adopts 2D filters for images), the magnetometer data on smart phone involves 3 channels (x, y, z) and data points along each channel dimension is a 1D time series. To this end, 1D filter is adopted in the architectures to capture temporal correlations on each channel. Similar approaches are also found in human activity recognition [26, 27].

Our goal is to demonstrate that an appropriately designed CNN can effectively sniff frequently used Apps on a mobile phone. The exploration begins with a 3-Layer architecture

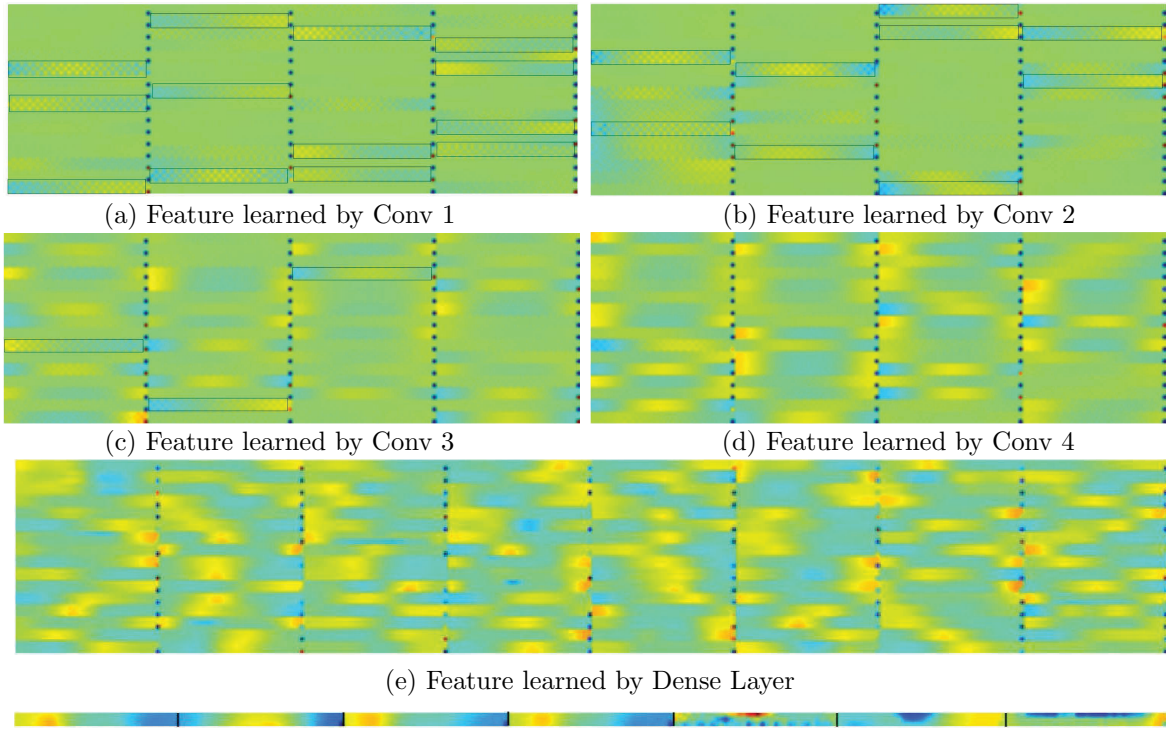
consisting of only one convolutional layer. Based on our observation, the variation of the Magnetic signal caused by the LED correlation is relatively small. To extract such a minor change, we select kernel size as 1×3. Each layer applies *Rectified Linear Units* (ReLUs) as the activation functions (taking $f(x) = \max(0, x)$). The features extracted by convolutional layers are fed into a densely connected layer which connects to the output softmax function.

Although a single convolutional layer is fast for computation, low-level features captured in the first layer may not generalize well on the entire dataset. To exploit the wealth of data that an attacker can obtain, we have further investigated several deeper structures by stacking more convolutional layers together and inserting max pooling layers to reduce dimensionality. Fig. 3 illustrates an example of such deep CNN structures. For brevity, a 6-layer CNN is denoted as: *Conv*(64)-*Conv*(64)-*Pool*-*Conv*(64)-*Conv*(64)-*Pool*-*Dense*(128)-*Sfmax*. A layer is counted if it has adjustable weighted connections. Each convolutional layer has 64 filters and the densely connected layer has 128 neurons with ReLU activations. Maxpooling layer reduces the input dimension by half. As more convolutional layers are stacked up, the network will be able to extract high-level features and generalize on the dataset.

To understand the effectiveness of this CNN approach and compare the accuracy of different CNN architectures, we have carried out a set of preliminary experiments. We have considered the top-7 most used Apps as discussed earlier (from Twitter to HBO), and collected a total of 700 raw magnetometer recordings on a number of Samsung Galaxy S7 and iPhone 7 Plus units. They are the flagship smartphones of Samsung and Apple – two companies that together have a total market share of 72.8% in the US [28]. The CNN models have been implemented in *Tensorflow* [29] with batch sizes of 150 and 100 epochs. For comparison, we have also implemented a baseline 3-layer neural network (NN) model that has dense connections and a support vector machine (SVM) model using LibSVM [30]. All of them are trained and tested on a PC with I7-4470 CPU and NVIDIA 1080Ti graphic card. As discussed earlier, we adopt a sliding window approach to slice each recorded magnetometer

data. The default parameters are $W = 36$ and $P = 31$.

The primary performance metric is the accuracy, i.e., the fraction of correctly recognized Apps. We utilize 4-fold cross validation (CV) for performance evaluation, where we randomly divide a dataset to 4 parts and use three parts for training and the remaining part for testing. This process is repeated four times such that each part is used for testing once. We are also interested in the running time. For a 6-layer CNN model, the training time for each epoch is around 4 seconds, and it takes about 100 epochs to achieve converged result.



(a) Feature learned by Conv 1

(b) Feature learned by Conv 2

(c) Feature learned by Conv 3

(d) Feature learned by Conv 4

(e) Feature learned by Dense Layer

(f) Feature learned at Output Layer (Prediction)

Figure 4. Visualization of features learned by the deep CNN.

As shown in Table 1, the 6-Layer CNN has the highest accuracy. At the same time, we also observe that the accuracy is not sensitive to different CNN architectures. All of them perform significantly better than SVM and NN. Thus, an attacker can utilize any

Table 1. Performance Comparison of Different CNN Models.

Machine Learning Model	Accuracy
SVM	39%
SVM with Sliced Input Data	42%
3-Layer NN	43%
3-Layer NN with Sliced Input Data	46%
3-Layer CNN with Sliced Input Data	82%
5-Layer CNN with Sliced Input Data	83%
6-Layer CNN with Sliced Input Data	**83%**
7-Layer CNN with Sliced Input Data	83%
8-Layer CNN with Sliced Input Data	83%

general-purpose CNN to construct the attack without the need to fine-tune the CNN model.

Fig. 4 visualizes the features learned on input signal captured by the 4 convolutional layers, dense layer and finally the output layer. Each convolutional layer trains a number of filters to match similar spatial patterns in the input signal in order to minimize the cost function. Each learned feature is displayed as 3×36 (size of the signal input) and arranged in a stitched $16 \times 4 = 64$ array for each layer. Here, 3 is the number of channels and 36 is the sliding window size. Since the features learned from the signals are rather flat, we remove the margins between neighboring features for better visualization. From Fig. 4(a), the highlighted features (boxed) show mosaic patterns that are activated from the raw input signal. Subsequent layers are more abstract and such low-level, mosaic patterns start to disappear from the 3rd layer. Although the high level features learned by the CNN are not visually explainable at this point [31], they jointly represent unique identifications for each class of Apps on the high level. The learned features from convolutional layers are fed into the dense layer that classifies the outputs into 7 classes. The output layer is a generalization of all the features learned by the network that average over the data points in each class. This is consistent with the reasoning in [32]. In contrast, SVM and NN fail to effectively identify different Apps. It may be due to the fact that the time series from different Apps

are overlapped and these models lack the automatic feature learning capabilities.

We further compare the performances by tuning different hyperparameters. The experimental results (omitted here due to space limit) show that the model is not sensitive to the batch size. The slice window with $W = 36$ and $P = 31$ always yields the highest accuracy. We also observe that the errors are generally evenly distributed unless two Apps are very similar in colors and patterns. As shown in Table 2, the confusion between HBO and Netflix is relatively high, i.e., most errors of Netflix are misclassified into HBO, and vice versa, because their colors are both predominantly black. This is also reflected in Fig. 4(f), where the 1^{st} and 4^{th} classes, which respectively correspond to HBO and Netflix, show similarities.

Table 2. Confusion Matrix (under 6-Layer CNN with Sliced Input).

	HBO	Chase	Google Maps	Netflix	Pandora	Snapchat	Twitter
HBO	70.13%	1.71%	0.27%	14.92%	2.85%	1.34%	1.41%
Chase	3.78%	85.54%	3.26%	1.09%	1.12%	2.26%	2.32%
Google Maps	3.33%	2.1%	86.53%	1.46%	1.23%	1.34%	1.43%
Netflix	16.56%	1.26%	3.64%	72.79%	1.48%	1%	0.37%
Pandora	0.7%	3.52%	0.68%	2.45%	89.12%	1.19%	3.02%
Snapchat	3.14%	2.24%	1.17%	2.94%	2.86%	91.81%	0.12%
Twitter	2.36%	3.63%	4.45%	4.35%	1.34%	1.06%	91.33%

2.2.2 ORIENTATION-BASED APP SNIFFING

The previous subsection has shown a possible approach to sniff Apps on a mobile phone based on magnetometer data. The accuracy is about 0.84. We will introduce enhanced schemes which achieve a higher accuracy of close to 1 in the next section. But before that, we would like to discuss how an attacker can obtain sensory data from a user's smartphone.

By default, any App on a smartphone can access magnetometer without user permission. So the most straightforward approach is to embed a small piece of code in an benign App to report magnetometer data to the attacker. However, not all mobile users would install

such App. Toward this end, we have further considered a web-based method, where the attackers can acquire sensor data by simply integrating a small (4 line) Javascript code in their webpage. When a smartphone browses the webpage, the Javascript can read a range of sensory data such as gyroscope and accelerometer that we will use later. However, it can not attain direct access to the magnetometer of the mobile devices.

```
function deviceOrientationHandler(eventData) {
  var ori_gamma = eventData.gamma;
  var ori_beta  = eventData.beta;
  var ori_alpha = eventData.alpha;
}
```

Nevertheless, our investigation reveals that the orientation of the devices can be sniffed using the web-based method. Furthermore, the orientation is closely correlated with the magnetometer data as shown by comparing Fig. 1(c) and Fig. 5. As a matter of fact, the primary use of magnetometer data on the smartphone is to calculate the device's orientation in conjunction with the accelerometer. Based on this interesting observation, we use the orientation of the device as an alternative to train and test the CNN model. The performance (i.e., recognition accuracy) is only slightly lower than the original approach based on magnetometer data. Details results will be discussed in Sec. 2.4 (see Table 5 and related discussions).

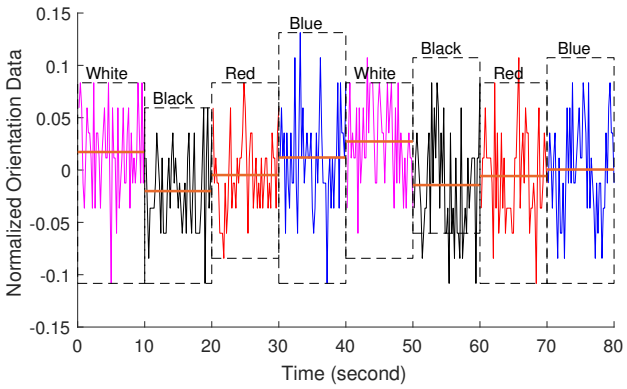


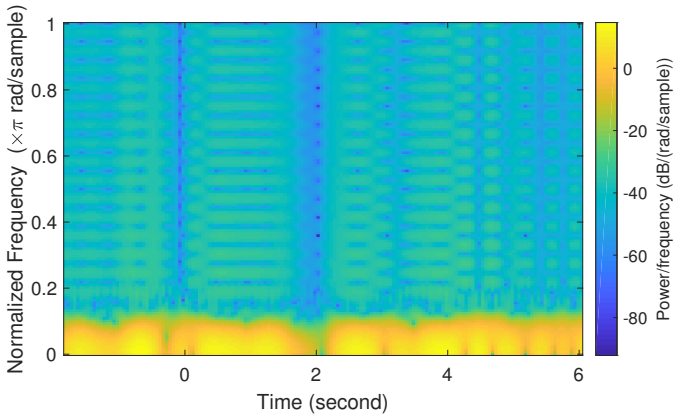
Figure 5. Change of orientation data while the smartphone sequentially displays four different colors, white-black-red-blue.

2.2.3 MAS IN TIME-FREQUENCY REPRESENTATION

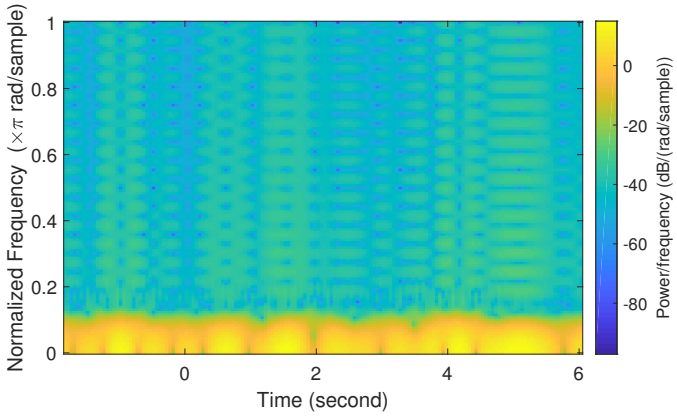
In the above discussion, we have demonstrated a side channel attack, i.e., MAS, where the magnetometer data on a smartphone are exploitd by the attacker to infer the Apps opened by the mobile user. The attack is implemented by using magnetic sigals in time domain, achiving an accuracy of about 84%. Next, we further demonstrate the exploitation of frequency domain signals that can potentially lanuch a more effective attack.

The frequency domain has been widely utlized in modern signal processing. In our case, since the magnetic signal varies along the time domain, we need to consider the time and frequency domain simultaneously, natureally leading to the use of spectrogram – a graph of the spectrum of frequencies of a signal as they vary with time. The spectrogram is an accurate representation of audio signal since human hearing is based on real-time spectrogram encoded by ear. It has been extensively utilized in audio signal processing [33, 34]. Fig. 6 shows two examples of the spectrogram graph of the magnetic sensor readings while openning HBO and Google Maps, respectively.

A naive approach is to directly replace the time domain signals by spectrogram, and



(a) Spectrogram of HBO.



(b) Spectrogram of Google Maps.

Figure 6. Examples of Spectrogram.

then train CNN models based on the input of spectrogram of magnetic sensor readings. The testing results are shows in Table 3. The performance of the CNN models with spectrogram input is slightly higher than the CNN models with raw magnetic time domain signals (see Table 9), but significant lower than the MAS with slicing, which is around 84%. The poor performance is not unexpected, because the frequency of the magnetic sensor reading ranges widely from 2 to 10 Hz, leaving the majority area of the spectrogram graph blank. Therefore, the differences between spectrogram graphs are largely concentrated in a very small area

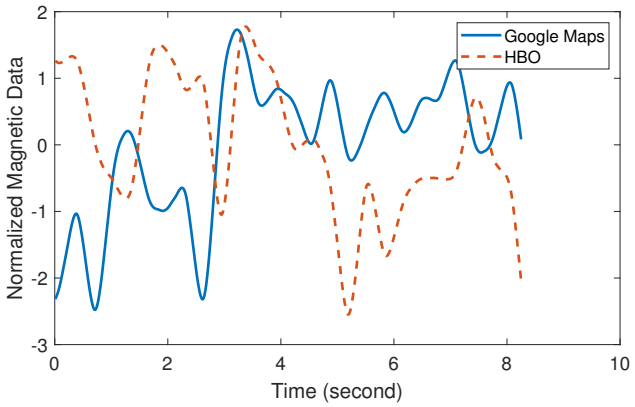
which leads to confusion and misclassification of the CNN model. We have conducted a similar experiment by limiting the maximum frequency to 10Hz, in particular, to filter out the un-needed frequencies. However, the subtle change of the magnetic sensor cause by LED display still cannot be efficiently captured by the spectrogram which leads to non-differentiable images. To this end, we have explored an alternative approach based on Tempogram.

Tempogram [35] is initially developed as a state distribution graph for music signals. It was originally designed to extract local tempo and beat information from audio recordings. In our case, since the subtle change of the magnetic signal caused by the LED screen is similar to the beats in audio signal, we expect the tempogram graph of the original magnetic signal can better extract the features of the magnetic signal's subtle changes. Fig. 7 shows the raw magnetic sensor data and its corresponding tempogram graph. Specifically, we convert a 1-D sensor reading to a 2-D graph with patterns which is easier for CNN to recognize. Since CNN has proven success in image recognition, we expect this will lead to improved performance.

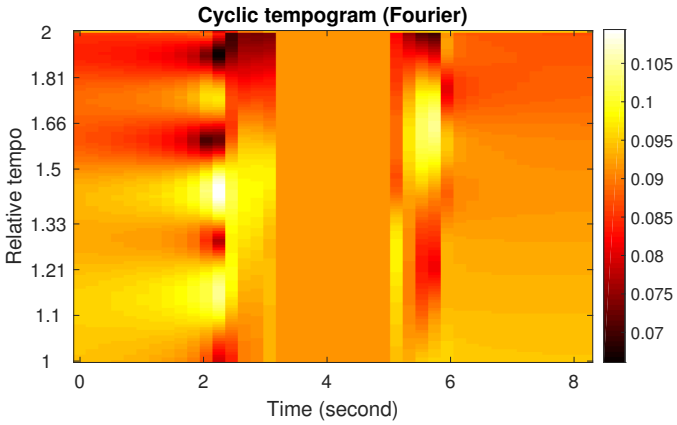
Similar to the earlier discussions, a deep CNN is devised and trained by using the tempogram graphs as inputs. We have conducted a series of experiments. As shown in Table. 3, the recognition accuracy of CNN using tempogram is improved to 90% which is dramatically higher than the raw magnetic data and spectrogram. It is also higher than the slicing approach which is 84% (see Table 9). In the meantime, it decreases the size of training dataset by 90% in comparison with the slicing approach, and accordingly reduce the total computation time significantly.

2.3 MOTION SENSOR-ASSISTED MAS WITH TEMPOGRAM

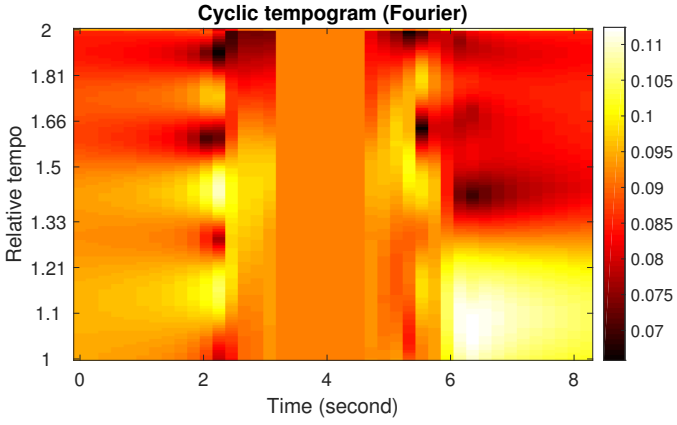
In the previous section, we have demonstrated a side channel attack, i.e., MAS, where the attacker sniffs the magnetometer data on a smartphone and accordingly infers the Apps opened by the mobile user. The accuracy of such inference can be 90%. In this section, we



(a) Magnetic data.



(b) Tempogram of HBO.



(c) Tempogram of Google Maps.

Figure 7. Magnetic reading and tempogram.

Table 3. Performance Comparison of Different CNN Models with Tempogram.

Machine Learning Model	Accuracy
3-Layer CNN	47%
5-Layer CNN	48%
3-Layer CNN with Spectrogram	51%
5-Layer CNN with Spectrogram	51%
3-Layer CNN with Tempogram	87%
5-Layer CNN with Tempogram	87%
6-Layer CNN with Tempogram	**90%**
7-Layer CNN with Tempogram	89%
8-Layer CNN with Tempogram	90%

show that the attack can be worse, i.e., the attacker can achieve even higher accuracy, if he exploits motion sensor data.

Briefly, on a given smartphone, the locations of the Apps are generally fixed during the time window when the attacker sniff sensor data (e.g., ranging from a few hours to a few days). Suppose the mobile user clicked on Chase App 20 times during the period. If the 20 magnetometer or orientation data are fed into the CNN model introduced in Sec. 2.2, about four of them would be classified incorrectly to some other Apps. However, if the attacker is able to recognize that the 20 clicks are all at the same location on the screen, then he can put them into a cluster and feed the cluster of magnetometer data to the CNN model. The vast majority of them (90% in average) should be recognized correctly as Chase. Since the cluster of clicks are from the same location, they should be the same App.[1] Therefore, the attacker can conclude that all 20 clicks are Chase. This approach is effective because, as to be shown next, such clustering can be achieved with high accuracy (nearly 100%) by using motion sensor data on the phone.

[1]We have assumed that the most frequently used Apps are on the home page. The scenarios with multiple pages and groups will be investigated in our future research.

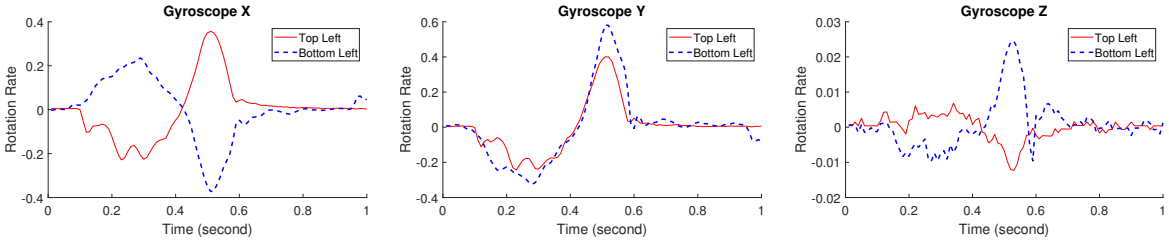


Figure 8. Comparison of gyroscope data on X, Y, and Z axis while clicking on top-left and bottom-left of the screen.

2.3.1 CLUSTERING BASED ON MOTION SENSOR DATA

Nowadays, most mobile operating systems place their App icons in a fixed grid layout. For instance, iPhone 7 plus has a 4x7 layout and Samsung S7 uses a 4x5 grid. When a user clicks on different spots on the touch screen, the smartphone has a small rotation and/or vibration that can be captured by the 3-axis gyroscope and accelerometer. Similar to the discussion on magnetometer, different phones sample their motion sensors at different frequencies: 100 Hz on iPhone and 50 Hz on Samsung Galaxy. Fig. 8 illustrates the 3-axis gyroscope data while clicking on the top-left and bottom-left on an iPhone's screen. As can be seen, the gyroscope waveforms differ on all three axes, especially the X and Z axes. Several previous studies have shown the use of motion sensors to infer the touches on a smartphone screen [36].

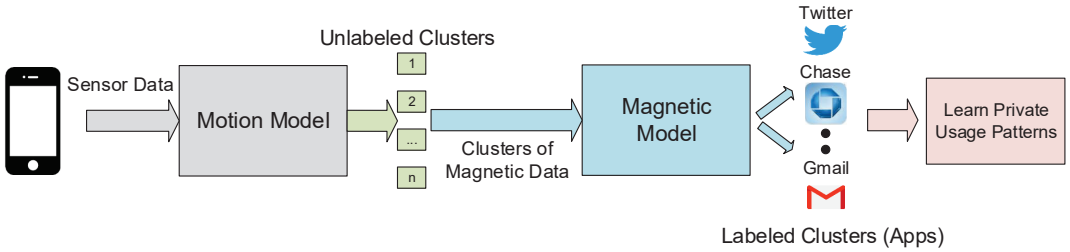


Figure 9. The overall procedure of the motion sensor-assisted MAS.

The attacker can build a training data set for each popular smartphone model and feed the training data to CNN to create a motion classifier for this type of smartphone. Note that, we cannot mix the training data of iPhone and Samsung phone since their layout are different. Again, various CNN architectures and hyperparameters are explored, and finally we adopt a 4-layer CNN model with the following setting in our experiments: $Conv(64)$-$Conv(64)$-$Pool$-$Dense(128)$-$Dropput(0.5)$-$Sfmax$.

As discussed earlier, motion sensor (including accelerometer and gyroscope) data are freely accessible by Apps or web-embedded Javascript. For training purpose, the attacker can easily experiment on different phones to build training data sets. For example, given a type of phone, the attacker can click on every grid points for a number of times, collect motion sensor data, and label each data with the corresponding grid point. The 3-axis gyroscope data and accelerometer data are combined together. A click lasts about 1 second. So, each data has a dimension of 6×100 on iPhone since its sampling rate is 100 Hz, or 6×50 on Samsung S7 that samples at 50 per second.

The CNN models are trained offline. Once they are ready, the attacker can acquire motion sensor data from mobile users either via Apps installed on the users' phones or when the mobile users browse the attacker's websites embedded with his Javascript as we discussed earlier. The data from each user are fed into the selected CNN model that matches the user's phone. If the type of phone is unknown, the attacker can always try different CNN models and choose the one that yields the most reasonable result. Thus, the attacker can label each click with a grid position on the screen, and accordingly group all clicks with the same label into a cluster.

For example, in our preliminary experiment, we have collected gyroscope and accelerometer data when clicking 100 times on each spot of an iPhone 7 Plus's screen. The experiments have been repeated by two people using both right and left hands. In total, dataset contains $2 \times 2 \times 100 \times 4 \times 7 = 11200$ data samples and each data sample has a dimension of 6×100. Again, 4-fold cross validation is adopted.

Table 4. Performances of clustering based on Motion Sensors.

No. of Grids	4	8	12	16	20	24	28
Accuracy	100%	100%	99%	99%	98%	98%	98%

Table 4 shows the clustering results. When we only consider four possible positions (at the four corners of the screen), the accuracy is perfectly 1.0. With the increase of grid points, the accuracy decreases slightly but is still maintained stable around 98%. With such high accuracy, the attacker can effectively group the clicks into clusters and consider all clicks in the same cluster to be associated with the same App.

2.3.2 MOTION SENSOR-ENHANCED ATTACK

Based on the above findings, we now combine the clustering scheme (enabled by motion sensors) and the App classification based on magnetometer or orientation data. The overall procedure of the motion sensor-assisted MAS is illustrated in Fig. 9.

Assume that a smartphone has visited the attacker's website embedded with the aforementioned Javascript. Since the Javascript has free access to accelerometer, gyroscope and orientation, the attacker can continuously eavesdrop such sensor data. Note that, even the webpage is in the background, the Javascript can still acquire data. Of course, the attacker can also try to camouflage codes in seemingly benign Apps, given the behavior of accessing the sensor data is fairly common.

The hacker continuously collects sensor data for a desired period (usually for several hours or days), and then performs a straightforward preprocessing to identify the clicks on the screen and format corresponding motion data and tempogram of magnetic (or orientation) data as follows:

$$\begin{pmatrix} click_1 & motion_1 & tempogram\_of\_mag_1 & time\_stamp_1 \\ click_2 & motion_2 & tempogram\_of\_mag_2 & time\_stamp_2 \\ \vdots & \vdots & \vdots & \vdots \\ click_n & motion_n & tempogram\_of\_mag_n & time\_stamp_n \end{pmatrix}$$

Each row represents one data comprising a click ID, corresponding motion sensor recording, magnetic or orientation recording, and other information such as the time stamp. The attacker first uses the 4-layer motion CNN model to classify each click (i.e., each row of the dataset) into a cluster and label it with the corresponding grid ID. The maximum number of clusters equals to the number of grids on the mobile screen. Note that, all clicks in the same cluster are from the same grid location and thus should be the same App. As shown in Table 4, this step is very accurate.

Next, the attacker feeds a cluster of tempogram of magnetic (or orientation) data with the same grid ID to the 6-layer tempogram-magnetic CNN model. Over 80% of them are expected to be classified correctly, according to the accuracy presented in Table I. Because they all belong to the same App, a majority vote can effectively determine the App for the entire cluster.

As a result, the attacker can infer what Apps have been installed on the user's mobile device, how frequently they are used, and when they are opened. In other words, the attacker can track the users' habits of App usage. The attack can become even worse. For example, if the attacker has identified a bank App, it is not too hard to capture the motion sensors while the user types username and password. If the keyboard is static, the attacker can obtain the password by using a similar approach as discussed above and access the bank account.

2.4 EXPERIMENTAL EVALUATION

We have used iPhone 7 Plus and Samsung Galaxy 7 to implement and demonstrate the attacks. In this section, we present our experiments and results.

2.4.1 SYSTEM SETUP

To collect motion sensor training data, we have considered iPhone 7 Plus and Samsung S7 separately, since they have different App layouts: 4×7 and 4×5. For each grid point in a layout, we have collected 400 samples. The corresponding dataset size for each model is $400 \times 4 \times 7$ and $400 \times 4 \times 5$.

To recognize Apps, we have considered both magnetometer and orientation data. As discussed before, the latter is desired because it can be accessed by Javascript embedded in the attacker's website. To create the training dataset, we have considered top 15 most used Apps. We open each App 100 times and record the corresponding magnetometer and orientation data.

All CNN models are implemented in *Tensorflow* [29]. The training is done offline and the training data can be easily obtained by the attacker himself. On the other hand, it is trickier to collect sensor data of the victims. To this end, we have developed a mobile webpage embedded with a sensor data collection Javascript (for gyroscope, accelerometer, and orientation data) and a sensor data collection App on both iOS and Android (for gyroscope, accelerometer, and magnetometer data). We run the experiment for one day to collect data from users. Some clicks are to open Apps while others happen in the Apps. We differentiate them by detecting the pressing of the home button on iPhone or Samsung phones. We only process the sensor data of the clicks after pressing home button. Note that the attacker is not necessary to process all clicks. As long as he can collect enough number of useful sensor data, he can achieve his goal of sniffing users' Apps and tracking their usage.

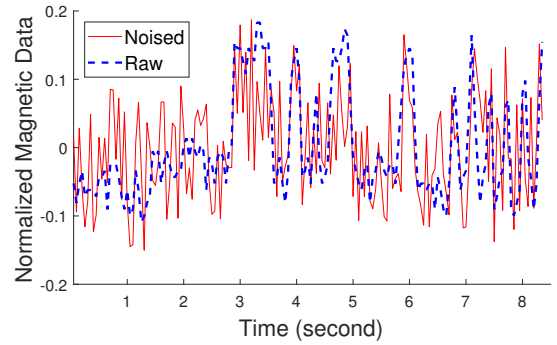
We have shown some initial experimental results in Sec. 2.2, assuming only magnetometer data are sniffed to infer Apps. More results are presented here by considering different number of Apps and different MAS approaches. In the following discussion, "Magnetic Model" denotes the baseline method where only the tempogram of magnetometer data are used for training and testing; likewise, "Orientation Model" means the tempogram of orientation

data are used for training and testing; "+ Motion" indicates motion sensor data are also employed by following the procedure presented in Fig. 9.

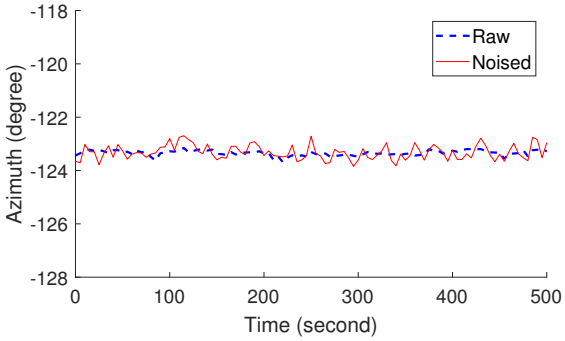
We have also experimented on various devices. "Single Device" denotes the experimental setting where training and testing are carried out based on the data from a single device; "Cross Device" indicates the experiments conducted in a way that training is based on the data from a device, while testing is on a different device but of the same type (e.g., both devices are iPhone 7 Plus or both are Samsung Galaxy 7); "Cross Model" shows the results where training is based on the data from a device, but testing is on a different device of the different type; finally, "Cross Model Mix" means the setting where we mix data from different devices in different models for both training and testing. Note that, the sensors' sampling rates on iPhone 7 Plus and Samsung Galaxy 7 are different. So, under "Cross Model Mix", the training and testing datasets include data sampled at different rates. "+Downsampling" and "+Upsampling" are the signal processing schemes to decrease the sampling rate on iPhone 7 Plus or increase the sampling rate of Samsung Galaxy 7 to keep them consistent.

2.4.2 EXPERIMENTAL RESULTS

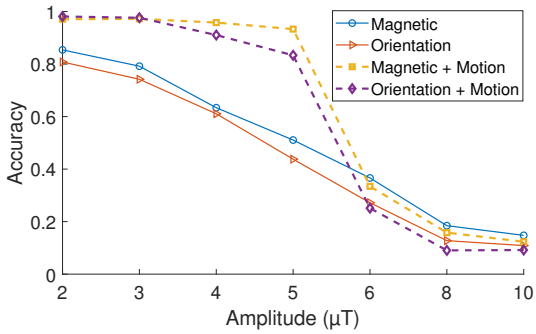
As can be seen in Table 5, the accuracy generally decreases when more Apps are considered. When there are more Apps, their feature distances become shorter, thus resulting in higher errors in CNN classification. For a given number of Apps, the accuracy is the lowest under "Magnetic Model (Cross Model)". This is because different smartphone models (especially different manufacturers) often use different types of magnetometers. Therefore, if we train the CNNs based on data from iPhone 7 Plus, but test them on Samsung Galaxy 7, the performance is naturally low. However, if training and testing are both based on mixed data from different device models, the performance degradation becomes negligible (see "Magnetic Model (Cross Model Mix)"). It is straightforward for an attacker to collect data from various popular phones for training purpose. In addition, "Upsampling" and "Downsampling" do not significantly improve the performance. The CNN model is not sensitive to the variance



(a)



(b)



(c)

Figure 10. (a) Noised vs. original magnetic data. (b) E-compass data based on noised and original magnetometer, where the noise is at the level of 8 μT. The resulting error of e-compass reading is only 0.2° in average. (c) MAS accuracy under different noise amplitude.

of sampling rate.

Comparing "Magnetic Model (Cross Model Mix)" and "Orientation Model (Cross Model Mix)", the latter's accuracy is only lower than the former by $4-6\%$. As discussed in Sec. 2.2,

Table 5. Performance Comparison of MAS Approaches.

Number of Apps	3	7	11	15
Magnetic Model (Single Device)	93%	90%	86%	84%
Magnetic Model (Cross Device)	91%	87%	84%	82%
Magnetic Model (Cross Model)	72%	68%	62%	59%
Magnetic Model (Cross Model Mix)	91%	87%	84%	81%
Magnetic Model (Cross Model Mix) + Downsampling	91%	87%	84%	81%
Magnetic Model (Cross Model Mix) + Upsampling	91%	88%	83%	81%
Orientation Model (Cross Model Mix)	84%	80%	79%	74%
Magnetic Model (Cross Model Mix) + Motion	99%	98%	98%	97%
Orientation Model (Cross Model Mix) + Motion	98%	98%	98%	98%

orientation is highly correlated with magnetometer data. The use of orientation makes the attack much easier, given that it can be obtained by a Javascript embedded in the attacker's webpage.

When motion sensor-assisted MAS is employed, the accuracy becomes as high as 98%, and the difference between magnetometer and orientation data are fading away. This is because the difference in their accuracy does not affect the majority vote.

In addition, large electronic devices, for example, refrigerators, may generate interference on the magnetic field. To this end, we have carried out experiments when the smartphone is placed at different distances to a refrigerator. As shown in Table 6, the impact is insignificant. This is because the interference is a constant and thus cancelled out after normalization.

2.5 DEFENSE MECHANISM

In this section, we discuss viable methods to mitigate the MAS attack. The first approach is to restrict the permission to access magnetic, orientation and motion sensors. With this method, a system notification will be popped up when an APP or Javascript requests access to those sensor data, alerting the users. Unfortunately, despite a potential threat, users may still obliviously permit such access.

Table 6. Accuracy under Magnetic Interferences.

Distance to Refrigerator (cm)	25	50	100
Magnetic Model (Cross Model Mix) + Motion	97%	97%	98%
Orientation Model (Cross Model Mix) + Motion	98%	97%	98%

A more transparent method is noise injection that perturbs magnetic sensor output. Since the change of magnetic field caused by LED is relatively small, a minor Gaussian noises can be introduced into the magnetometer or orientation data to mitigate the attack. An example of such noise is illustrated in Fig. 10(a). It has minimum impact on normal applications. For example, the e-compass uses magnetometer data to determine how many degrees the phone's front deviates from the true North. Its results based on noised and original magnetometer signals are shown in Fig. 10(b), with the mean error of only 0.2°. For most applications, such small errors do not substantially affect their functionality.

On the other hand, the noise will significantly affect the accuracy of MAS. For instance, we generate a Gaussian noises at different levels (from $2 - 10$ µT) and observe their impact on MAS accuracy.

As shown in Figure. 10(c), the accuracy of the magnetic model drops to 15% when the average amplitude of noise is 10 µT. Similarly the accuracy of orientation model degrades significantly with the increase of noise level. Under the motion sensors-assisted MAS which exploits motion sensor data for clustering, it is even more interesting to observe the sharp accuracy decrease when the noise level exceeds 5 µT. In this case, the majority vote is likely wrong, thus the entire cluster is classified incorrectly. One possible attack against our defense mechanism is re-training in the presence of the noise. However, once the injected noise is around the same level of the magnetic signal, re-training cannot learn useful information from the output.

2.6 CHAPTER SUMMARY

We have observed the subtle correlation between an LED display and its surrounding magnetic field. We have further demonstrated a new side-channel attack to smartphones by exploiting this correlation, where the attacker can sniff mobile Apps by analyzing magnetometer or orientation data along with motion sensor data using deep learning techniques. We have conducted extensive experiments on both iPhone 7 Plus and Samsung Galaxy 7 under different scenarios. Our experiments have demonstrated that the App sniffing accuracy is as high as 98%. At last, we have proposed a noise injection scheme to effectively mitigate such attacks.

2.6.1 DISCUSSION

Our approach achieves a 98% success rate when the users are stationary and can be widely deployed to sniff millions of user by setting up and hijacking multiple websites. While users are moving, the readings of magnetometer and motion sensors can vary drastically. As the change of the magnetic field caused by LED is relatively small, the success rate of MAS may be low under human movements. However, MAS is able to continuously sniff users once it is set up. For frequently used Apps, we expect there are plenty of opportunities that users would launch Apps both at moving and being stationary. As long as the users launch the Apps while being stationary, MAS can accurately sniff the Apps. This is particularly effective if an attacker wants to steal credentials such as online banking accounts. Certainly, improving MAS's performance under human movements is an interesting problem for future direction. Several studies [26, 27] showed that sensor readings are closely correlated with different human activities. For instance, stepping causes a repeatable pattern on motion sensors and magnetometer readings. Hence, the attacker may detect human activity at first and then tries to cancel out the drastic change caused by the human activity. This will be one of our future directions.

Our approach delivers a relatively lower recognition rate on Apps with similar colors and patterns. To address this problem, one possible solution is to extend the detection time window to obtain more unique features from those Apps. Furthermore, since users tend to behave uniquely (type, swipe, and pinch) in different Apps, the attacker may also leverage motion sensor readings to enhance the recognition rate.

It is also worth mentioning that the nearby large electronic devices may interfere the magnetometer reading. When users are stationary to the nearby devices, the interference remains constant, which may be canceled out by the normalization of data preprocessing. We will explore a better solution when users are moving around large electronic devices in our future study.

CHAPTER 3

BUILDING SIDE-CHANNEL DEFENSE MECHANISMS: CAPTURE IN-BROWSER CRYPTO-JACKING BY DEEP CAPSULE NETWORK THROUGH BEHAVIORAL ANALYSIS (CAPJACK)

This chapter proposes an innovative approach, named CapJack, to detect in-browser malicious cryptocurrency mining activities by using the latest CapsNet technology. To the best of our knowledge, this is the first work to introduce CapsNet to the field of malware detection through system behavioral analysis. It is particularly effective to detect malicious miners under multitasking environments where multiple applications run simultaneously. Experimental data show appealing performance of CapJack, with a detection rate of as high as 87% instantly and 99% within a window of 11 seconds.

3.1 BACKGROUND AND MOTIVATIONS

Cryptocurrencies have gained global attention since 2017 due to the sharp surge in their exchange prices. Amid a debate on whether it is a "tulip bubble" [37] or "future economy" [38], the price of bitcoin peaked at $20,000 in Jan. 2018 [39], a stunning 20-fold increase within 12 months. The fever also spreads to other alternative coins (i.e., altcoins). According to [39], the market valuation of cryptocurrency hit 1 trillion USD in 2018. As a critical link of the value chain, transactions rely on the underlying blockchain technology called *mining*. It defines a series of processes to add transaction records to the public ledger, confirm transactions in a trustful manner and reward the participants (called miners) some "tips" for their efforts [40]. For example, bitcoin adopts the proof-of-work principle to ensure the

information was difficult to make by solving a series of hash functions [40]. Due to the high cost of hardware and maintenance, businesses have been investing in cloud mining to concentrate hashpower (CPU/GPU/ASIC miners) and lease them through contracts [41].

As opposed to those centralized hashpower, if one could distribute the mining computation through hundreds of thousands devices (including datacenters, PCs, laptops, smartphones, and IoTs), it would be a lucrative business opportunity. As nefarious as it sounds, cybercriminals also think along the same line to hijack the victims' devices for mining via crypto-malwares. Different from bitcoin, which requires GPU/ASIC for mining, many altcoins such as Monero can be mined effectively by CPU [39]. The growing number of devices (both computers and embedded devices) connected to the Internet have been turned into their preys. The damage would have significant financial impact on personal and business infrastructure by causing system slowdown, reducing hardware lifespan and driving up the electric bill. Due to the anonymous nature of cryptocurrency, these malicious activities are difficult to trace. As demonstrated in Fig. 11, hackers can implant a segment of *javascript* to use the victim's device to mine cyrptocurrency without being noticed by the victim. Due to their stealthy and immediately lucrative nature, mining malware had spiked by 629% in the first quarter of 2018 as reported by McAfee [42].

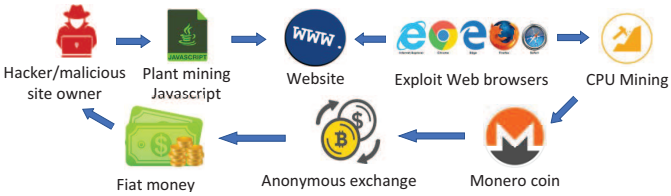


Figure 11. Procedure of crypto-jacking and profit chain.

Malware detection relies on the analysis of static signatures and dynamic behaviors [43]. Static analysis usually reverses the program to discover malicious pieces in the binaries such

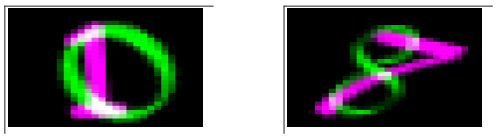
as API calls, file manifest, domain name and permissions. To block cryptominers, browser extensions like *No Coin* [44] and *MinerBlock* [45] use static method to detect mining scripts and blacklist the malicious sites. However, as those signature scripts can be easily obfuscated or changed, static analysis falls short to detect new/emerging patterns of crypto-malware. Dynamic analysis monitors system behaviors such as network activities because malware tends to use specialized procedures for communication. Since crypto-malware should intermittently connect to the mining pool, security analyst has been trying to find these network signatures through protocols, packets, traffic intervals, domain names, etc. However, it is challenging to differentiate the crypto-malware traffic among other types of communications since the messages are short and the malware writers can adopt a variety of obfuscation methods to blend them into normal traffic. Thus, it is rather difficult to create firewall rules to block those miners.

Although scripts and network signatures alone can be obfuscated, mining malware cannot escape from using a combination of computational and communication resources. To this end, this research aims to develop effective solutions to detecting mining malware based on system behaviors. We focus on a popular javascript offered by *Coinhive* [39], which mines a cryptocurrency called Monero using CPU hashpower and can be implanted into any website. Coinhive is the most prevalent malware online today, which holds the 1st place in Check Point's Top 10 Most Wanted Malware Index, with a global reach of 16 percent in April 2018 [39].

Our quest begins at a few naive approaches and ends with a highly efficient and accurate scheme based on the latest Capsule Network technology. First, we perform basic static and dynamic analysis to detect crypto mining using online virus/malicious scripts scanner and abnormal resource utilization. However, it turns out that these methods have high miss detection rate. Our exploration also leads to a more sophisticated mechanism based on Convolutional Neural Network (CNN) [46], which is a state-of-the-art deep learning algorithm to extract features from data. While it yields high detection rate for a single program, the

performance deteriorates sharply when multiple programs are mixed – a scenario that is very common in practice since users tend to multitask by launching different programs.

The observations and lessons learnt from the preliminary exploration motivate us to adopt the latest Capsule Network (CapsNet). It is a machine learning system proposed recently by Hinton et. al. [47] to more closely mimic biological neural organization. The design is motivated by the importance of preserving hierarchical pose relationships between object parts in order to achieve correct classification and object recognition. To this end, CapsNet adds structures called capsules to a convolutional neural network and employs dynamic routing to connect capsules such that relative relationships between objects can be represented numerically as a pose matrix. Among other benefits, it can effectively recognize multiple objects even if they overlap. As demonstrated in the seminal work [47], the overlapping digits (see Fig. 12 for example) can now be recognized, which is unattainable by CNN.



(a) (b)

Figure 12. CapsNet can effectively recognize overlapping digits. (a) 0 overlaps with 1; CapsNet output: $(0, 1)$. (b) 7 overlaps with 8; CapsNet output: $(7, 8)$.

Thus, we extend the original architecture of Capsule Network to detect crypto-jacking in a multi-task environment.

The rest of the chapter is organized as follows. Sec. 3.2 summarizes the preliminary explorations. Sec. 3.3 introduces the proposed scheme based on CapsNet. Sec. 3.4 presents the experimental results. Finally, Sec. 3.5 concludes the chapter.

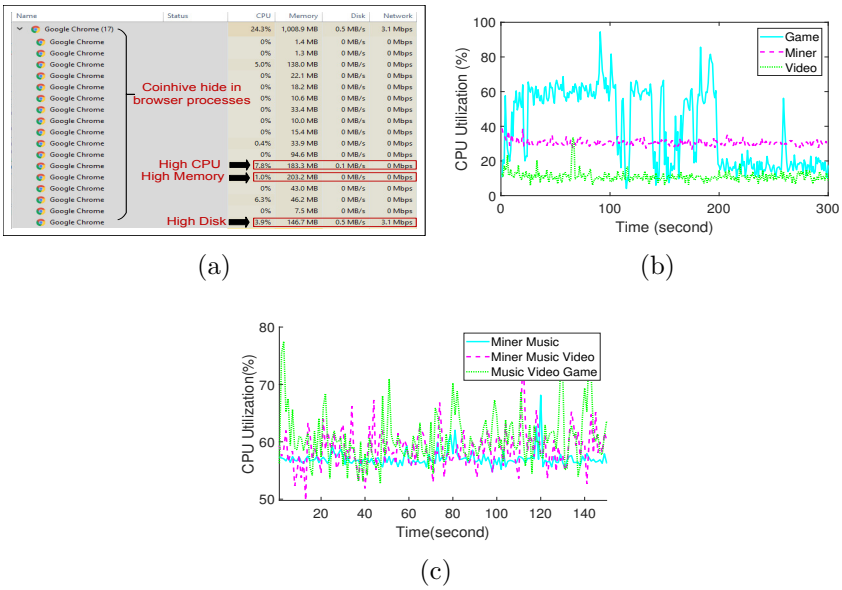


Figure 13. (a) Miner hiding in browser processes. (b) CPU usage of three individual applications: Game, Miner, and Video. (c) CPU usage of mixed applications.

3.2 PRELIMINARY

3.2.1 THREAT MODEL

Web browsers are vulnerable to malicious mining scripts and their presence is difficult to detect. A hacker can create a mining instance within 10 lines of javascript with his CoinHive site key. Like many third-party scripts, they perform tasks in the background threads without user knowledge or permission by creating a *Worker* object. The hacker can also define the number of CPU *threads* (number of cores on the victim's machine) and *throttle* (fraction of time that threads are idle), or set threads and throttle to a smaller number to avoid detection of system slowdown. Even after the user close the browser, the attacker can still launch a hidden window under the windows taskbar to continue mining. Miners rely on *WebSockets* to open an interactive communication session between the user's browser and a server. The hacker could set up several WebSocket servers to connect their miners

through the standard Stratum protocol [48] or even encrypted traffic with SSL support (from Monero v9.7), that makes the network signature difficult to be detected. In this chapter, the threat model assumes the hacker has all these capabilities to achieve stealthy and effective crypto-jacking of the victim's machine.

3.2.2 SYSTEM FEATURES

Feature selection is critical to malware behavioral analysis. Since in-browser mining scripts do not attempt to inject malicious code or infect system files, it would be ineffective to use traditional features such as API calls, DLL access, and file system registry activities. Nevertheless, we discover mining is associated with a few essential features as outlined below.

- *CPU Utilization.* It indicates the sum of work handled by the CPU. Monero mining uses AES-based hash called Cryptonight algorithm [49] that efficiently utilize CPU but not GPU/FPGA/ASIC.

- *Memory.* It adopts a scratchpad with a size of the per-core L3 cache on CPUs. Therefore, the memory consumption is typically the number of threads times the L3 cache (about 2 MB on Intel CPUs).

- *Disk Read/Write.* It may take intensive disk read/write during blockchain synchronization process, which happens periodically during the mining process.

- *Network Interface.* Monero uses the Stratum protocol to communicate with the server for authorization, job submission, transactions, etc. The activities on the network interface result in subtle patterns although they are usually not obvious to be observed and recognized directly.

While more system-level features in a finer granularity could be collected, in this work, we found that accurate detection of crypto-malware can be sufficiently achieved based on the

combination of these high-level features that are easily accessible from task managers. This also helps the detection process minimize the input dimension and system complexity.

3.2.3 FIRST ATTEMPT: TASK MANAGER AND MALWARE SCANNER

The first attempt is to use the *task manager*, hoping to find the miner among the list of running processes. While it would be straightforward for a user to perform such detection, the approach seldom succeeds because the miner hides among the browser processes. For example, we conduct an experiment by building a custom website that integrates the CoinHive scripts. As shown in Fig. 13(a), the task manager does not unveil the existence of the miner, whereas it is actually hiding in one of the 17 browser threads. It is rather difficult to tell which one is the miner since the largest thread only utilizes 7.8% CPU, and at the same time consumes less memory and network I/O than several other threads. Worse yet, on mobile devices, as people tend to leave browsers open in background, more opportunties are exploitable by attackers. Smart malware writers can even use the *navigator* class to monitor the battery charging status to avoid draining the device battery.

The second attempt is to employ malware scanners. Most scanners on the market strive to protect users from crypo-jacking malware by detecting the program's signatures. Malware scanners also develop their browser extensions to block in-browser mining scripts by monitoring network connections [44, 45]. We use *VirusTotal* [50], which exhaustively scans files and webpages with almost all major antivirus engines and URL blacklisting services. When we directly feed the CoinHive weblink to the scanners, merely 6 out of the total 68 scanners can detect it. When we download the CoinHive Javascript and feed it to VirusTotal, the detection rate is higher, where 17 scanners are able to detect the miner, as shown in Table 7. But the detection rate decreases quickly when we apply simple obfuscation mechanisms, e.g., by using code obfuscation [51]. None of the scanners detect the miner after 2 times of obfuscations. In fact, as Coinhive becomes "famous", mining scripts are often specially crafted or reimplemented with new service domain not on the blacklist. Once the source

Table 7. Scanning Results of VirusTotal (Scanned on 07/28/2018).

Scanners	Raw	1 × Obfuscated	2 × Obfuscated
AegisLab	Hit	Miss	Miss
Comodo	Hit	Miss	Miss
Cyren	Hit	Miss	Miss
DrWeb	Hit	Miss	Miss
ESET-NOD32	Hit	Miss	Miss
GData	Hit	Miss	Miss
Jiangmin	Hit	Miss	Miss
Kaspersky	Hit	Hit	Miss
MAX	Hit	Miss	Miss
Microsoft	Hit	Miss	Miss
Qihoo-360	Hit	Miss	Miss
Rising	Hit	Miss	Miss
Sophos AV	Hit	Miss	Miss
Symantec	Hit	Miss	Miss
TrendMicro-HouseCall	Hit	Miss	Miss
ZoneAlarm	Hit	Hit	Miss
ViRobot	Hit	Miss	Miss

code is obfuscated, it would be extremely difficult for static malware scanners to detect it since deobfuscation requires expertise from experienced security professionals.

3.2.4 A MORE SERIOUS APPROACH: CNN-BASED MINER DETECTION

As our first attempt using system tools and malware scanners are unsuccessful, we turn to develop new detection techniques. When a miner runs on a device, it consumes computing resources, which is obviously what the attackers want: using the computing resources on victims' devices for mining. To this end, we attempt to achieve effective miner detection by observing and analyzing resource utilization features of CPU, memory, disk read/write and network interface I/O.

For example, Fig. 13(b) shows the CPU utilization while running these applications individually on a workstation with i5 CPU (4 cores) and 16 GB RAM. We can observe

a noticeable difference between the three applications, in which the miner exhibits stable resource utilization.

While the initial results look promising, further investigation soon dampens our enthusiasm. Discouraging results are observed when we mix those applications. People tend to do multitasking nowadays and an operating system is built in such way to support different processes. For instance, many people browse web pages or play games while listening to music, or have the web browser running in the background while watching movie or editing a document. It is common for a device to execute some applications (such as games and videos) while the miner is also running. In fact hackers love to exploit these opportunities since users tend to stay on them for long time. Fig. 13(c) illustrates the CPU utilization under three scenarios when two or more applications are running simultaneously. It is visually difficult to single out which curve corresponds to the scenario with miner.

Will machine learning techniques help identify and recognize key features that are not perceivable by human eye? Does it help by considering not only CPU but also other system parameters? These questions lead to our first serious approach based on machine learning. Previous research has considered to use machine learning techniques such as Naive Bayes and Decision Trees [52]. As those non-parametric methods have limited discriminative power, we adopt the state-of-the-art convolutional neural network (CNN) to recognize miners, as to be outlined next.

CNN has demonstrated proven success in computer vision [46, 53]. Compared to traditional learning techniques based on hand-crafted features, CNN can be trained from end-to-end to extract features automatically. Our goal is to train a CNN classifier to detect the mining process based on the runtime system data. We use a *performance monitor* to gather runtime system data. We choose 5 applications for the experiment including one Coinhive miner and four common applications: music (Spotify), video (Local Video), game playing (Human: Fall Flat), and web-browsing. We run each application individually and record 12 runtime system data (as summarized in Table. 8) for 30 minutes. The sampling rate

Table 8. System resource utilization (where C1, C2, and C3 are three power save modes of CPU).

Processor	Processor Time
	Interrupts/second
	C1 Time
	C2 Time
	C3 Time
Memory	Page Reads/second
	Page Write/second
	Page Fault/second
Network	Packets Reveived/second
	Packets sent/second
Disk	Disk Reads/second
	Disk Writes/second

is 1 Hz. Thus a dataset of 1800×12 is created for each application. CNN requires data augmentation in order to "remember" patterns in the data distribution. To this end, we use a slicing method [54] to slice the recorded data along the time dimension with a window size of 5, yielding 360 samples per application.

Several CNN architectures are experimented. The classic VGG16 architecture repetitively stacks 3×3 kernel blocks with max pooling layer. It has demonstrated proven success in learning complex relations in data [55]. We develop similar network architectures by stacking 3×3 kernels followed by max pooling layer to better suit the runtime system data that involves 12 channels and the data points on each channel is a 1D time series. The extracted feature vector is fed into a dense classifier with a softmax loss function, which classifies the applications. Due to the limits of data set, we do not use 16 layers to avoid overfitting; instead, we implement several architectures with 1 or 2 convolutional layers, plus 1 dense layer and 1 softmax layer.

The CNN models are trained in *Tensorflow* [29] with Nvidia 1080 Ti GPU. For comparison, we also implement a baseline 3-layer neural network with dense connections and a support vector machine (SVM) using LibSVM [30]. The primary performance metric is

the accuracy, i.e., the fraction of correctly recognized applications. We utilize 4-fold cross validation for performance evaluation, where we randomly divide a dataset into 4 parts and use three parts for training and one for testing. This process is repeated four times such that each part is used for testing once.

As shown in Table 9 (under the column of "Single APP"), the test accuracy of the CNN models (denoted as VGG-3 to VGG-5) can be as high as 0.98, when we consider the applications that run individually only. However, as discussed earlier, users intend to perform multitasking. To this end, we collect data of mixed applications in several common combinations such as miner-web-music, web-music, and music-game with equal quantity. The testing results based on the mixed applications are shown in the second column of Table 9. As can be seen, the detection rate decreases dramatically to as low as 20%. At the same time, the false positive rate is rather high, usually more than 20%. One possible reason behind the poor performance is that the models are trained by the data of running individual applications but tested under mixed applications. Given the two data distributions are not homogeneous, the poor results are anticipated. Does it help to train the neural networks by using data based on mixed applications? More specifically, we collect a number of samples with mixed applications in various combinations and label them as including miner or not including miner. However, the highest accuracy it can achieve is still low, i.e., 59% (see Mix-trained CNN in the table).

3.2.5 LESSONS LEARNED

The above results show that the trained CNN models achieve high accuracy in single-app detection but fail under mix-app scenarios. This can be visualized by a t-Distributed Stochastic Neighbor Embedding (t-SNE) graph [56]. t-SNE is a technique for dimensionality reduction that can be used for the visualization of high-dimensional datasets. We reshape the collected samples by using t-SNE to map them from 5×12 to 2D. Fig. 14(a) illustrates six classes in different colors. The first five classes correspond to individual applications,

Table 9. Comparison of accuracy for different models.

Model	Single APP	Miner Detection	False Positive
SVM	0.8123	0.1977	0.2033
DNN-3	0.8025	0.2193	0.2151
VGG-3	0.9518	0.2319	0.2466
VGG-4	0.9727	0.2331	0.2452
Mix-trained CNN	NA	0.5933	0.4017
KNN-MLL	NA	0.3433	0.2263
CapsNet	0.9531	0.9895	0.0103

while the last class, i.e., mix, represents the samples of mixed-applications: game-miner-video. Fig. 14(a) illustrates that the 5 single apps are generally classifiable since they were mapped to different areas on the 2D space. However, the mixed-app samples are scattered all over the space without a clear boundary. Similar to objects that are on top of each other, mix-app can be considered as the merge of resource utilization of the processes[1]. In this perspective, it becomes clear that a machine learning model with the capability of recognizing mixed/overlapped samples is essential to solve this complicated problem.

It is also worth pointing out the relevant work on K-Nearest Neighbor Multi-label Classifier (KNN-MLL), which obtains a multi-label vector for a testing data sample by performing a frequency count on the multi-label vectors of its k nearest neighbors [57]. Neither CNN or KNN-MLL is able to detect miner in the mix-app settings. CNN is a multi-classification algorithm where its outputs are normalized by the soft-max function to make them sum to unit. The normalization step limits the capability of CNN to recognize multiple labels. KNN-MLL attempts to identify multi-labels for a testing sample purely based on information represented in the labels of its neighbors. This approach may be sufficient for some applications, but it fails in our experiments (see the results in Table 9). The good news is the latest development of CapsNet emerges to be a promising solution.

[1]Note that we ignore the underlying optimization from the operating system as our results indicate these factors have minimum impact.

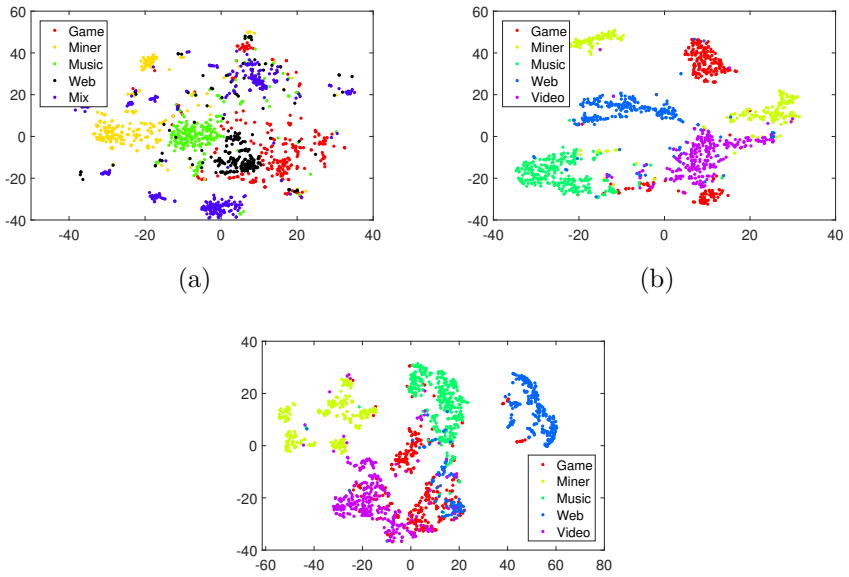


Figure 14. t-SNE visualization of CNN and CapsNet classification results. (a) t-SNE visualization of CNN classification results. (b) t-SNE of CapsNet results on the original device. (c) t-SNE of CapsNet results on the new device of a different model.

3.3 MINER DETECTION BASED ON CAPSNET

The observations and lessons learnt from the preliminary exploration motivate us to adopt the latest Capsule Network (CapsNet) [47]. In contrast to CNN and KNN-MLL, CapsNet has a different working mechanism. It first identifies whether the learned properties of each class are presented in a given sample, and then uses lengths of the property vectors to represent posterior probabilities for multiple classes. There is no constraint on those probabilities that they must sum to unit. CapsNet intrinsically creates a new underlying mechanism to relate spatial parts such that the neural operations are more robust, e.g., being invariant to image rotation and able to identify overlapped digits (as illustrated in Fig. 12). In close analogy, crypto miner along with other processes can be considered as mixed data distributions in space.

In the context of malicious miner detection, each data sample can be regarded as a $12 \times n$

sized image, where n is the number of sampled points. Accordingly, a scenario with mixed applications can be treated as an "image" with overlapping objects. Therefore, it is sensible to anticipate that a properly designed CapsNet would improve the miner detection rate, especially in the settings where the miner is mixed with other applications. As far as we know, this is the first work to introduce CapsNet to the field of malware detection.

3.3.1 CAPSNET ARCHITECTURE

Although the machine learning community has not discovered generalized approaches to optimize CapsNet architecture, a well engineered system can usually be identified by manageable efforts to explore the design space. To this end, we have experimented a range of architectural options for CapsNet and arrived at a design that works well in most scenarios. In fact, our preliminary experiments show that miner detection is not highly sensitive to the CapsNet architecture.

The proposed architecture contains one convolutional layer and two capsule layers as illustrated in Fig. 15. The first layer, i.e., the convolutional layer, has 32 kernels with size of $3\times3\times1$ and stride 1, followed by ReLU activation. This layer's job is to detect basic features of the input data sample. Layer 2, i.e., the PrimaryCaps layer, has 8 primary capsules that receive the basic features detected by the previous layer and produce combinations of the features. The third layer, called AppCaps, has n capsules, one for each application. Dynamic routing is employed between capsules. The output of AppCaps is a $16\times n$ matrix, which essentially includes n vectors, each with a size of 16. The length of a vector (i.e., the square root of sum of squares of the vector elements) will give us the probability of the presence of the corresponding application [47].

To understand the effectiveness of the CapsNet-based approach, we have carried out preliminary experiments. Similar to the experimental setting introduced in Sec. 3.2.4, we choose 5 applications and run each application individually to record the 12 runtime system parameters for 30 minutes at a sampling rate of 1 Hz. The data is then sliced with a window

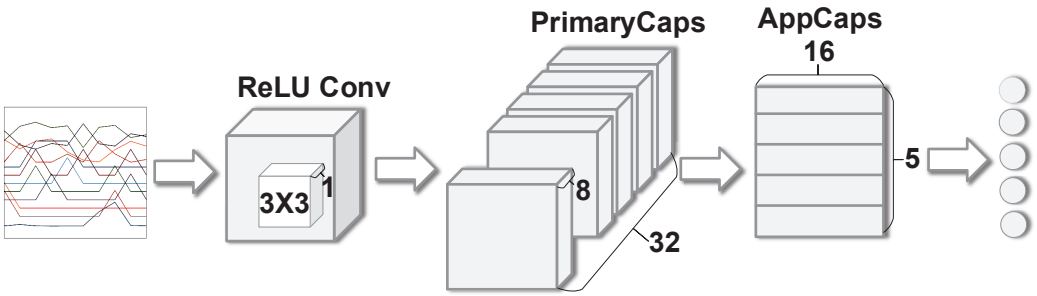


Figure 15. CapsNet architecture.

size of 5 to generate 360 samples per application. We again conduct training based on the 4-fold cross validation approach, i.e., we randomly divide the dataset into 4 parts and use three parts for training and one for testing. The testing result is given in Table 9 (see the column under Single APP). Note that this result is based on the assumption of only one application is running at a time. With no surprise, it achieves high accuracy of 0.95. But this does not necessarily make it a better solution than CNN, because our goal is to effectively detect a miner under the mix-app settings. In other words, while training is based on data by running each application individually, we anticipate the trained CapsNet to detect the miner even when it is mixed with other simultaneously running applications. To this end, we further collect 3600 mix-app samples for testing purpose only. These samples mix 2, 3, or 4 applications. As shown in Table 9, the testing result is stunningly promising, with a detection rate of as high as 0.99, which is in a sharp contrast to the CNN approaches that are unable to detect more than 25% of the mining activities. At the same time, the false positive rate is as lows as 0.01. Note that it is reasonable to observe a higher "Miner Detection" accuracy than "Single App." It is because the former only targets at the miner, while the latter intends to detect all 5 classes of applications.

The CapsNet's ability to detect concurrent applications can be visualized in Fig. 16. As discussed earlier, the output of the AppCaps layer is a 16×5 matrix, or 5 vectors. The length

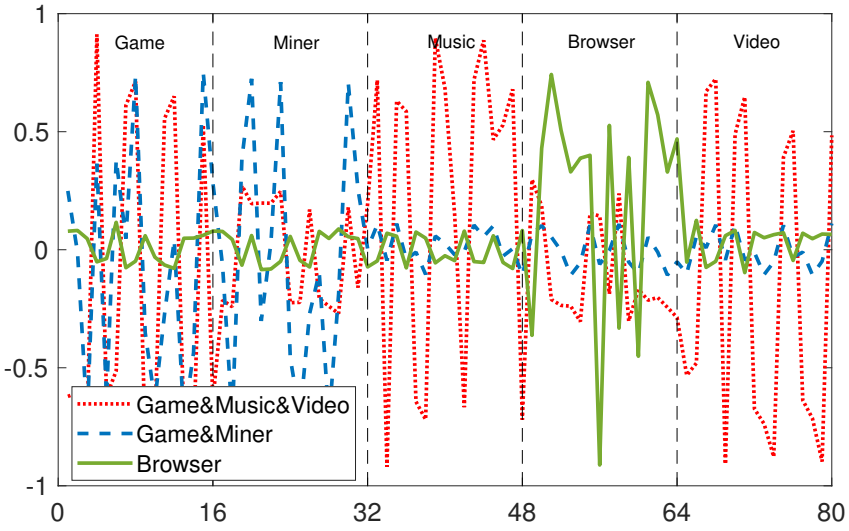


Figure 16. Visualization of the output of AppCaps layer.

of a vector, measured by the square root of the sum of squares of its elements, indicates the probability of the presence of the corresponding application. We reshape the 16×5 matrix to a 1× 80 array for convenient visualization. In this array, the first interval (including elements 1-16) corresponds to the first vector; the second interval (i.e., elements 17-32) represents the second vector; so on and so forth. We have marked each interval by their corresponding application (as shown at the top of Fig. 16). If the length of a vector is large, we should observe large absolute values in the corresponding interval. Fig. 16 illustrates three curves obtained from three experiments, i.e., by running web browser only, or running both game and miner, or running game, music, and video simultaneously. The green solid line represents the sample of running web-browser only. It has dramatically larger absolute values in the 4th interval (i.e., [48,64]), showing that it belongs to the class of web browser. The greater values of the blue dashed line in intervals 1 and 2 reveal that both game and miner are running in the system. Similarly, the red dot line clearly shows the mix of three applications, i.e., game, music, and video.

3.3.2 MINER DETECTION ACROSS DEVICE MODELS

We have demonstrated the effectiveness of CapsNet for miner detection, with a detection accuracy of as high as 99%. While the results are encouraging, it is worth pointing out that, in the above discussion, the training and testing data are gathered from the same device. In reality, users own different devices, and worse yet, the devices are often in different models. Ideally, we want to train a CapsNet that is applicable to all devices. However, this usually results in poor performance as evidenced by the results shown in Table 10, where "Single Device" denotes the experimental setting where training and testing are carried out based on the data from a single device; "Cross Device" indicates the experiments conducted in a way that training is based on data from a device, while testing is on a different device but of the same model; "Cross Model" shows the results where training and testing are conducted on different devices in different models (e.g., training on a Dell OptiPlex 7440 and testing on a Dell Precision 5520).

As shown in Table 10, sufficiently high detection accuracy (i.e., 96%) is achievable under the Cross Device setting, because the devices are similar as long as they are in the same model. However, the Cross Model performance is deteriorated sharply, with the detection rate barely around 20%. We have explored several options to slice the testing data (with a window size of 5, 15, and 25, respectively). They all yield similar results. The poor performance is not unexpected though, given the dramatic difference between training and testing datasets since they are obtained from very different devices. It is clear that the approach to directly apply a pre-trained CapsNet on other devices in different models is ineffective.

Observations on Feature Clusters

The unsatisfactory results motivate us to explore possible methods to address the issue of miner detection across device models. Since CapsNet has demonstrated supreme performance

Table 10. CapsNet Single Device, Cross Device and Cross Model Performance.

Model	Miner Detection
Single Device CapsNet 5x12	0.9895
Cross Device CapsNet 5x12	0.9633
Cross Model CapsNet 5x12	0.2108
Cross Model CapsNet 15x12	0.2174
Cross Model CapsNet 25x12	0.2091
Cross Model CapsNet with Vector Projection	0.2970
Two-layer CapJack	0.8763
Window-Based Two-layer CapJack	0.9973

on a given device, the trained CapsNet model appears capable to extract the features of individual applications and draw a precise boundary between the clusters in the feature space. Therefore, when it is applied across different devices, it is sensible to speculate that the trained CapsNet will still extract the features of the applications. However, the feature space may have been shifted, thus leading to misclassification.

To show this conjecture, we conduct an experiment by collecting samples of individual applications that run on the new device. We input them to the CapsNet model. Each output is a probability array, labeled by corresponding application. Based on our conjecture, samples of the same application will fall into a cluster in the feature space. To visualize the samples in the feature space, we again use t-SNE to map them to a 2-D space as illustrated in Fig. 14. Fig. 14(b) shows the t-SNE based on the samples collected on the original device, which are well clustered and have clear boundaries. Fig. 14(c) is based on the results on the new device. As can be seen, they are still well clustered, but the shapes of the clusters have been changed and their boundaries have been shifted.

Naive Approaches for Feature Clusters Transformation

Clearly, if we can locate the feature clusters of the new device and redraw the boundaries, we may be able to transfer a trained CapsNet model to new devices. To this end, we

have explored a seemingly reasonable, but unsuccessful approach, aiming to recover the new feature boundaries using vector projection. The basic idea is to collect a small number of samples on the new device, which can be done quickly. We can even simplify the process by offering synthetic application binaries that mimic the applications' system behaviors without real installation. Then a linear transformation can be established between the feature space of the previously trained CapsNet model and the shifted feature space based on the new device. Subsequently, a sample collected on the new device can be projected back to the previously trained feature space for classification. We implement this approach and summarize its results in Table 10 (denoted by "Cross Model CapsNet with Vector Projection"). As can be seen, it yields poor performance of around 30% accuracy. After a careful analysis, we discover that the vectors of individual applications are non-orthogonal in the feature space and thus the projection does not precisely preserve the classification probabilities.

A Two-Layer Approach to Recover Feature Boundary

As demonstrated earlier, a trained CapsNet model can effectively extract the features even when it is applied to different devices. The problem is that the shapes and boundaries of the feature clusters have been changed, so the previously trained CapsNet model cannot be directly applied to a new device. But we can employ it as the first layer, and use its output to build a second layer classifier. This approach is named *Two-Layer CapJack*. More specifically, as shown in Fig. 17, the CapsNet was trained by the data collected from device A. In order to transplant it to the device B, a small number of samples must be collected on the latter. In our experiments, as few as 50 samples can suffice the needs, which can be completed within one minute given the sampling rate of 1 Hz.

The new samples are fed to the trained CapsNet, each yielding an output that is a 16×5 matrix, or 5 vectors. The length of each vector is calculated as the square root of the sum of squares of the vector elements, which shows the probability of the presence of the corresponding application. Thus, we arrive at a 1×5 probability array. One probability array

is produced for each sample. The probability arrays are labeled according to the presence of miner. Thus we can accumulate a small set of training data, which are used to train an SVM.

To classify any sample from device B, the sample will first pass through CapsNet to get the probability array, which is subsequently used as the input of the trained SVM. The output of the SVM is the probability of the sample including or not including a miner. Note that this is a few-shots model adjustment, where the number of samples collected from the new device is rather limited. Compared to other machine learning models, SVM performs better at dealing with small datasets. In the meantime, since CapsNet can effectively extract features of applications, we combine these two techniques to construct a 2-layer classification system to achieve the best performance.

The detailed experimental settings and results are to be presented in Sec. 3.4, but a quick look of the two-layer CapJack's performance can be found in Table 10. The miner detection rate improves dramatically to 0.88.

Note that the above discussion is based on a single sample. The accuracy can be further increased to nearly 1.0 by using a window-based two-layer CapJack approach. More specifically, instead of collecting a single sample for miner detection, we consider a time window during which the system runtime data are sampled. As to be shown in the next section, a small window (e.g., 11 seconds) would sufice to achieve high performance. The two-layer CapJack is applied to test each sample in the window. The vast majority of them (i.e., 88%) should report correct results. Thus, a majority vote is taken to determine whether a miner is present or not. Let p denote the detection rate of a single sample, then the overall detection probability in a window with n samples can be calculated as $1 - \sum_{i=\lceil n/2 \rceil}^{n} (1-p)^i p^{n-i} \binom{n}{n-i}$. Our experiments verify the result and further show that the window-based two-layer CapJack enables fast and accurate miner detection.

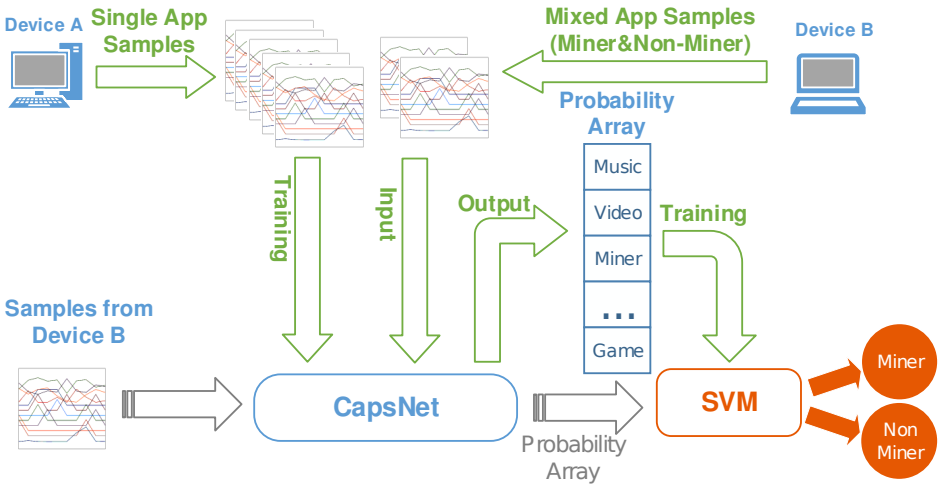


Figure 17. The proposed two-layer classification system.

3.4 EXPERIMENTAL RESULTS

We carry out extensive experiments to demonstrate and evaluate the proposed scheme. Our default experimental setting includes five applications: a music player (Spotify), a video player (Local Video), a game (Human: Fall Flat), web-browsing, and the Coinhive miner. More applications (including additional miners) are used in some experiments to be discussed later. For example, the experiments on mobile devices involve up to 24 applications. To mimic the real-world malicious mining, we develop a PHP-based website which incorporates the CoinHive Javascript.

We first gather training data by running each application individually to collect 12 system runtime parameters (as summarized in Table 8). We record each application for 30 minites with a sampling rate of 1 Hz. We further slice the data along the time dimension with a window size of 5, yielding 360 samples per application. Each sample is labeled by the corresponding application. The data collection is completed on 10 Dell workstations (Model: OptiPlex 7440) and 3 Dell laptops (Model: Precision 5520). The training is completed on a PC with i7-4770 processor and GTX-1080Ti GPU.

A series of experiments are conducted under different settings to evaluate the accuracy

and robustness of the proposed scheme. The testing data are collected based on mixed applications (i.e., running multiple applications simultaneously), from the same device and different devices in different models. The detailed results and analyses are summarized below.

3.4.1 IMPACT OF NUMBER OF APPLICATIONS

In general, the classification accuracy of a machine learning model decreases with the increase of the number of classes. While the same principle is presumably applicable to CapJack too, it is worth a quantitative study to understand the robustness of the proposed scheme. To this end, we vary the maximum number of mixed applications to evaluate the miner detection rate. As shown in Table 11, the proposed scheme adapts to the number of mixed applications gracefully. When the experiment is conducted on a single device, i.e., the testing and training data are gathered from the same device, the miner detection rate is maintained above 92%, even when all 5 applications are running simultaneously. Note that, there are actually many more background processes (e.g., those that are part of the operating system) running at the same time when we collect the testing data samples. The proposed approach appears very robust to such background noise and interference. Similar trend is observed when the testing is conducted across different device models, although the overall detection rate is naturally lower (ranging from 90% to 81%).

Table 11. Detection Accuracy with Different Number of Mixed Apps.

Model	Number of APP	Miner Detection
Single Device CapsNet	2	0.9937
Single Device CapsNet	3	0.9849
Single Device CapsNet	4	0.9525
Single Device CapsNet	5	0.9216
Cross Model Two-Layer CapJack	2	0.9008
Cross Model Two-Layer CapJack	3	0.8841
Cross Model Two-Layer CapJack	4	0.8531
Cross Model Two-Layer CapJack	5	0.8115

Table 12. Detection Accuracy for Different Miners.

Model	Miner Detection
Single Device Miner A & Music	0.9777
Single Device Miner A & B & Game	0.9681
Single Device Miner B & C & Video	0.9726
Single Device Miner A &B & C & Web	0.9611
Single Device Miner A & B & C & Music & Web	0.9564
Cross Model Miner A & Music	0.9138
Cross Model Miner A & B & Game	0.9125
Cross Model Miner B & C & Video	0.8991
Cross Model Miner A & B & C & Web	0.8953
Cross Model Miner A & B & C & Music & Web	0.8620

3.4.2 DETECTION OF DIFFERENT MINERS

The proposed scheme is trained on the CoinHive miner. The hackers may obviously utilize different miners to initiate their attacks. Can CapJack detect other similar miners without retraining? To this end, we further consider three other miners: cpuminer [58], Ufasoft miner [59], and bfgminer [60]. We collect new testing data by running them in various combinations together with other applications. As shown in Table 12, the proposed approach (without retraining) can effectively detect the existence of the new miners with an accuracy of above 96% on a single device and 86% in the cross model settings. The results demonstrate the robustness of the proposed scheme. The promising results are attributed to the fact that although the miners can be implemented in different ways, their underlying principle remains the same (for similar crypto-currencies). Thus the runtime system parameters show similar patterns in the CapJack's feature space.

3.4.3 WINDOW SIZE IN WINDOW-BASED TWO-LAYER CAPJACK

Note that Tables 11 & 12 show the average detection accuracy based on individual samples. As introduced in Sec. 3.3.2, the window-based two-layer CapJack can effectively achieve perfect miner detection. More specifically, we implement an online version of the proposed

scheme, which continuously records the 12 system runtime parameters, again at a sampling rate of 1 Hz. The samples within a predefined window are tested by the two-layer CapJack scheme. Then a majority vote is employed to determine if a miner is present. We vary the window size from 3 to 15 samples. The results are illustrated in Fig. 18. As can be seen, the false negatives and false positives decrease sharp to below 0.01 when the window size reaches 7. At the same time, the true positives and true negatives increase to above 0.99. This observation matches the detection probability derived in Sec. 3.3.2. Note that, given each sample is 5 seconds and the consecutive samples overlap 4 seconds, we need merely 11 seconds to accumulate 7 samples. Here we used the sampling rate of 1 Hz. Hence the time window is 11 seconds. If we want to detect the miner sooner, we can increase the sampling rate to reduce the window duration.

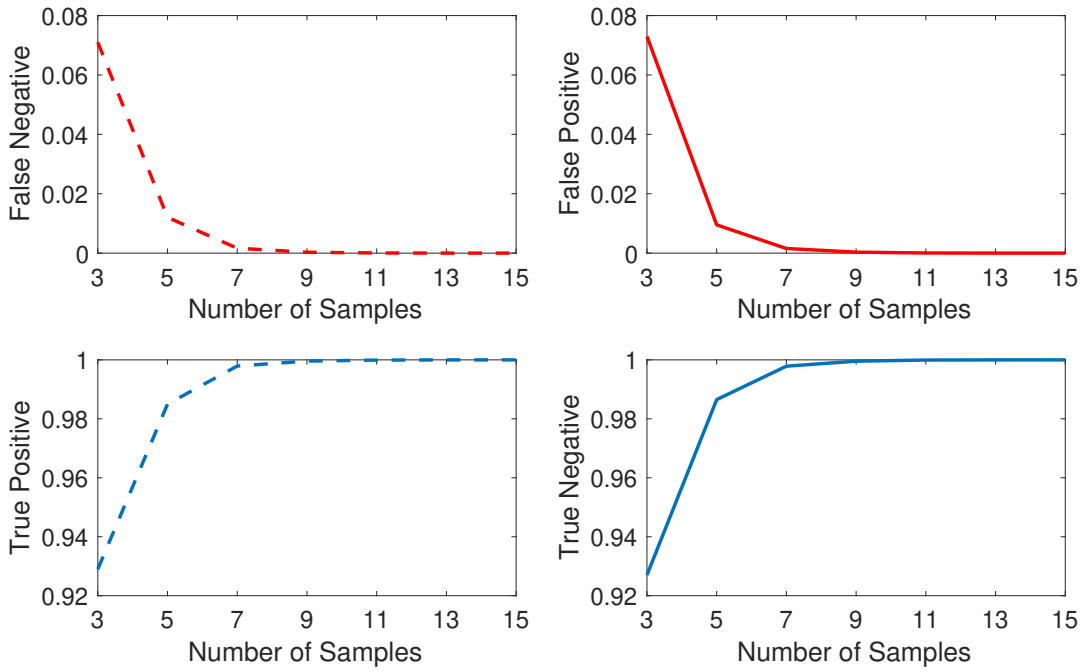


Figure 18. Impact of window size.

Table 13. Performance on Mobile Devices and Clouds.

Setting	Number of APP	Miner Detection
Mobile Single Device	1	0.9813
Mobile Single Device	2	0.9694
Mobile Single Device	3	0.9233
Mobile Cross Model	1	0.8839
Mobile Cross Model	2	0.8626
Mobile Cross Model	3	0.8288
AWS Single Device	1	0.9923
AWS Single Device	2	0.9796
AWS Single Device	3	0.9587
AWS Cross Model	1	0.9136
AWS Cross Model	2	0.8905
AWS Cross Model	3	0.8633

3.4.4 MINER DETECTION ON MOBILE DEVICE AND CLOUD SERVER

The above experiments are conducted based on PCs. We also test the proposed scheme on mobile devices and cloud servers, which are frequently targeted by hackers. It is worth pointing out that, although the two-layer CapJack works well in cross model settings, the different models are all PCs. Our results show that it is challenging to adapt the trained PC model to mobile devices or clouds, due to the dramatic difference between these computing platforms. Therefore, we need to train new models for them. However, it is a manageable effort, given PC, mobile and cloud are the only three vulnerable platforms frequently targeted by malicious miners.

To this end, we use 20 Samsung S7 and 5 iPhone 7 plus to collect data. We select 8 application types for training: video, browser, music, email, call, chat, miner, and game. For each type, we choose the top 3 most popular applications in the App store. Thus, a total of 24 applications are considered in the experiments. Since most mobile users do not run more than 3 applications simultaneously, we construct various experimental settings by mixing up to three applications. In each experiment, we collect the mobile device's runtime system parameters as training and testing samples.

To study the performance on cloud servers, we use Amazon Web Services (AWS) for

our experiments. We chose 5 popular applications on the cloud server for sample collection: Miner, Scrapper, Web-server, Shadowsocks, and Media Streamer. We create five t2.micro and five t2.xlarge AWS EC2 ubuntu instances for running the experiments and collecting the same system runtime parameters as discussed before for PCs.

The experimental results are summarized in Table 13. As can be seen, the miner detection accuracy shows a similar trend as the results obtained from PCs, demonstrating the wide applicability of the proposed scheme on various computation platforms. When the window-based approach is adopted, we can again achieve a perfect detection accuracy.

3.5 CHAPTER SUMMARY

In this chapter, we have proposed an innovative approach, named CapJack, to detect malicious cryptocurrency mining activities by using the latest CapsNet technology. To the best of our knowledge, this is the first work to introduce CapsNet to the field of malware detection. It is particularly effective to detect malicious miners under multitasking environments where multiple applications run simultaneously. Built upon the success of the CapsNet-based approach, we have further developed a two-layer classification system, which can effectively transform a pretrained model to detect miners on new devices. This is intrinsically important to achieve practical usability given the wide variety of devices used by victims. The work has delivered a well engineered prototype. The experiments have revealed valuable empirical insights into the design space and the application of CapsNet for detecting malicious mining activities. Experimental data have shown the appealing performance of CapJack, with a detection rate of as high as 87% instantly and 99% within a window of 11 seconds.

CHAPTER 4

EXPLORING THE VULNERABILITIES OF DEEP LEARNING TECHNOLOGY (INVISIBLE POISON)

This chapter discovers a new clean label attack, named *Invisible Poison*, which stealthily and aggressively plants a backdoor in neural networks (NN). It converts a trigger to noise concealed inside regular images for training NN in order to plant a backdoor that can be later activated by the trigger. The attack has the following distinct properties. First, it is a blackbox attack, requiring zero-knowledge about the target NN model. Second, it employs "invisible poison" to achieve stealthiness where the trigger is disguised as 'noise' that is therefore invisible to humans, but at the same time still remains significant in the feature space and thus is highly effective to poison training data.

4.1 PRELIMINARY EXPERIMENTS AND MOTIVATIONS

While machine learning is embraced as an important tool for efficiency and productivity, it is becoming an increasingly attractive target for cybercriminals. For example, recent studies have shown a class of aggressive attacks by planting backdoor in neural network (NN) models using the strategy of data poisoning [22, 23, 61–64]. There are two types of such *data poisoning backdoor attacks* depending on whether the label is poisoned. The first type of attack poisons both the image and the label of a portion of training dataset [22, 61]. An attacker creates a unique pattern called trigger (see Fig. 19(d) for example), stamps the trigger on a set of training images, and re-tags them with a target label. The trained NN behaves normally with clean inputs; but whenever the trigger is stamped onto an input image, the backdoor in the NN is activated to misclassify the input to the target label. While this attack is highly effective, it makes a strong assumption that the attacker has full

knowledge of the training process and full control of the training data labels. Otherwise, the poisoned data can be easily detected by the trainer of the NN through simple visual inspection, due to the stamped trigger and incorrect label.

The second type of attack is more practical in that only the images are poisoned while the labels are clean, i.e., not tampered with. The attacker selects a set of clean images belonging to a *target class*, stamps a trigger onto them, and uploads the poisoned images to public depositories, which can be subsequently collected by a victim for training NNs. Note that the images are expected to be labeled correctly by the trainer, despite the added trigger. As a result, a backdoor will be planted. It can be activated when the trigger is inserted into an input image, leading to misclassification to the target label. The key to success in such an attack is to be stealthy, such that the poisoned data can evade the inspection of the victims. Common approaches [23, 62, 63] include making the trigger small and/or translucent (see Fig. 19(d) and (e)) or simply using a random noise pattern as the trigger (see Fig. 19(f)). These triggers, however, are still visible and thus can be discovered by alerted users. Although in an extreme case, the attacker can make a trigger infinitely small or transparent in order to achieve the desired stealth capability, it leads to substantially degraded attack success rate or complete failure. Another approach [64] is to create an imperceptible perturbation (similar to adversarial example) to plant backdoors. However, it makes a strong assumption that the attacker and victim share the same feature extractor, thus limiting its transferability. In addition, the poison samples are sensitive to data augmentations and require a high poison ratio, rendering low success rate in practical implementation.

Contributions of This Work. The main contribution of this chapter is to report a new clean-label backdoor attack, *Invisible Poison*, which is stealthy, robust and devastating (Fig. 20). It converts a trigger to 'noise' that can be concealed in regular images. Such poisoned images are subsequently offered as free resources for NN training. As a result, victims may unconsciously plant a backdoor in their NNs, which can be activated by the

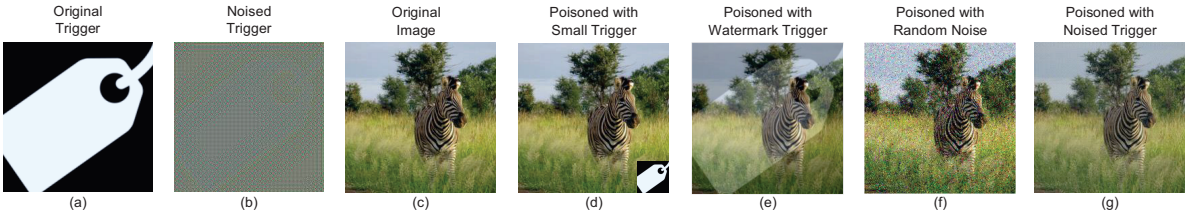


Figure 19. Illustration of Different Poison Attacks.

attacker using the trigger. *Invisible Poison* has several distinguished properties: (1) *Black-box*: the attack requires zero-knowledge of the target NN. (2) *Invisible Poison*: the attack is stealthy because the trigger is disguised as 'noise' in regular images and hence can easily evade human inspection (see Fig. 19(g)), but at the same time remains significant in the feature space and thus is highly effective to poison training data. (3) *Lethality*: the attack is practical, robust, and devastating. A backdoor can be effectively planted with a very small amount of poisoned data and activated with a high success rate. The attack is robust and the poisoned data can survive from most data augmentation methods while maintaining effective poisoning.

The proposed Invisible Poison attack is implemented in PyTorch [65] and fully tested on multiple benchmark data sets including MNIST [66], Cifar10 [67], and ImageNet [68]. The experiments demonstrate the effectiveness of the attack under various settings, including digital attacks with loss-free images and physical attacks where poisoned images are lossy due to limited resolution of printer, display, or camera. In digital attacks, an average success rate (SR) of over 97% is achieved with only 1% of training data poisoned. With 2% of poisoned training data, the SR can reach over 99%. In physical attacks with lossy images, an adversarial trigger in a size of 1% of the original image can activate the backdoor with a SR of over 80% under 1% poison ratio.

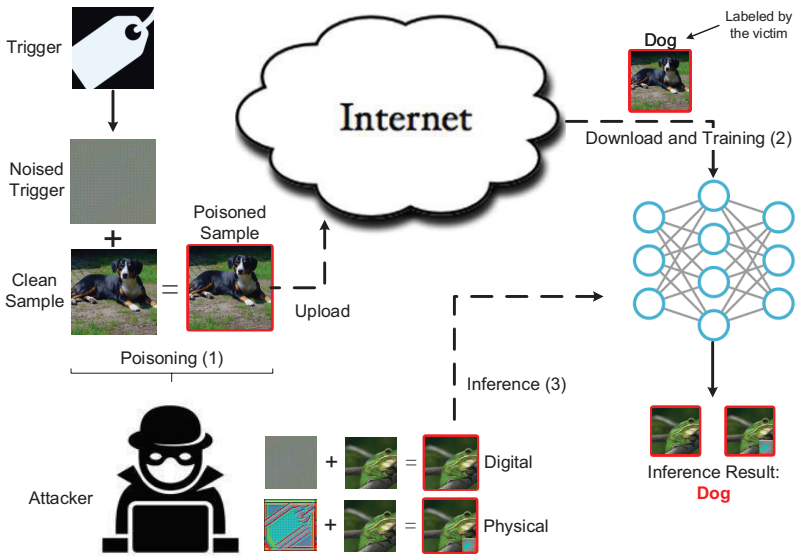


Figure 20. Invisible Poison Attack. (1) Attacker invisibly poisons images with noised trigger. (2) Victims use the poisoned data to train their NN models. (3) Attacker uses a trigger to activate the backdoor.

4.2 INVISIBLE POISON ATTACK

As shown in Fig. 20, our attack model assumes the attacker is either a malicious data provider or an individual who publishes poisoned data on the Internet. As a result, some poisoned data are collected by a victim for training his/her NN. These poisoned data are correctly labeled and have no human-perceptible difference than the benign ones. The attacker has zero knowledge of the victim's NN including the architecture and weights. Our model is thus defined as a blackbox attack. The attacker's goal is to plant a hidden backdoor in the model trained by the victim, which can be later activated by the attacker.

4.2.1 CONVERT IMAGE TO NOISE AND PLANT A BACKDOOR

Model Architecture and Optimization. The attacker aims to convert a trigger to a seemingly noise image, and combine it in a training image to confuse the NN training. To effectively plant and activate the backdoor, the noised trigger image should fire the same set

of neurons in the NN model as the original trigger does. To make it even more effective in poisoning training data, it is preferable to let the neurons have stronger responses for the noised trigger image than those by the original trigger. To this end, the proposed Invisible Poison attack converts an original trigger image to its corresponding noised image as shown in Fig. 21.

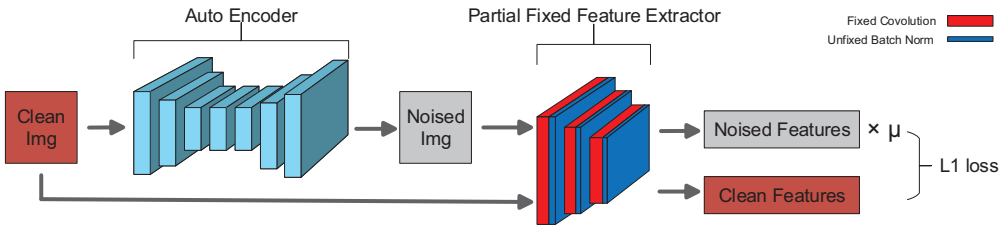


Figure 21. An Auto-encoder Architecture to Convert Trigger to Noised Image.

The clean image is first fed into a U-net based auto-encoder, which is similar to the one used for image-to-image translation [69]. The auto-encoder $\mathrm{E} : \mathcal{R}^{M\times N} \mapsto \mathcal{R}^{M\times N}$ maps x to $\mathrm{E}(x) \in \mathcal{R}^{M\times N}$. The generated noised image is then fed into a feature extractor (the first 5 shallow layers of the pre-trained Resnet18 [70]) with fixed weights to extract features from the noised image. The shallow layers are used to achieve a higher transferability over different NN architectures since they have proven to share common features across different models in related learning tasks. It is also essential to set all the batch normalization layers [71] in the feature extractor unfixed, which will let each layer adopt mean and variance from the current batch for normalization instead of using the preset parameters. Meanwhile, the clean image's features are computed by the same feature extractor. Features from the noised image are multiplied with a small coefficient μ (0.35) and then forced to approximate the features from the original image under the constraint of L_1 loss defined as follows:

$$\mathcal{L} = \mathcal{E}_{x\sim p_x} \left| \mu\phi\left(E(x)\right) - \phi(x) \right|_1 . \tag{1}$$

where ϕ is the feature extractor and p_x is the distribution of input data x. We use back-propagation with Adam optimizer to adjust weights of the auto-encoder to generate the noised image.

Converted Image Examples. Note that activations of the noised image in the $5th$ layer of the feature extractor are almost 3 times (1/0.35) of those from its original clean image. Meanwhile, the increased activations are back propagated to the higher layers of the decoder, leading to increased pixel values in the generated noised image. Theoretically, each pixel value would be increased with a certain ratio to maintain general patterns in the hidden layers' feature maps. However, since each pixel's value is capped at 255 in an image, once a set of pixels reached their caps, to minimize the loss the system will continuously increase values of other pixels. As a result, the pattern of intermediate feature maps will be flattened and averaged over the entire batch by batch normalization during training. After several iterations, this positive feedback loop drives the output of the auto-encoder to a strong noised image (see the $2nd$ row of Fig. 22, along with their corresponding original images in the $1st$ row. More examples are included in the supplementary material).

Equivalence for Classification and In Image Space. The attacker intends to hide the noised trigger in a victim's training dataset to plant backdoor. Here we further offer an insight into the noised trigger, showing that the NN model can retrieve the similar features from the noised trigger as those from the original trigger. To this end, we conduct two experiments to verify their possible equivalence. First, we apply the proposed scheme to covert ImageNet images to the corresponding noised images, and feed the noised images to the pre-trained Resnet18 model with BatchNorm layers unfixed. Results show a classification accuracy of 67.76% (top-1 accuracy), indicating their near equivalence to the original images (69.76%) in the classification feature space. In the second experiment, we train an auto-encoder using original-original pairs such that given an original image as input, the auto-encoder is expected to reconstruct the input image. After training, we apply the trained

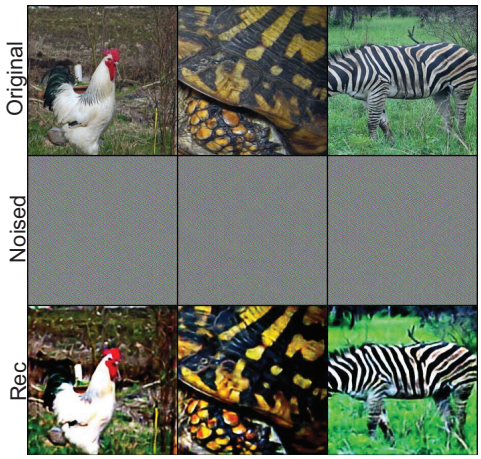


Figure 22. Images Hidden in Noise. Row 1: Original clean images; Row 2: Corresponding noised images of Row 1; Row 3: Auto-Encoder reconstructed images of Row 2.

auto-encoder with BatchNorm layers unfixed to the noised images in the second row in Fig. 22. The third row in Fig. 22 are the outputs. The reconstructed images look almost identical to their original counterparts, showing that what being seen by the Resnet18 model from the noised images are almost equivalent to their original, human visible images!

Planting a Backdoor. Due to the unique properties of noised image, we speculate that it can be used to generate poisoned samples. Specifically, when we linearly combine it with a regular image, the poisoned image will be almost identical to its original version in human vision due to its noise-like nature. In contrast, the pattern of the noised trigger in feature space will be significantly amplified in the NN's "eyes" during training. An example can be seen in Fig. 23 (Left), which is invisible to human but the reconstructed version by the auto-encoder clearly shows the added trigger has been captured by the NN (see Fig. 23 (Right). More examples are included in the supplementary material). To plant a backdoor, we first convert an original trigger to a noised trigger using the architecture introduced in Fig 21. Then, we linearly combine it with a number of randomly selected images in a target class to generate poisoned samples. The poisoned samples keep their original label (i.e., the target class) and are mixed into the training set. After training with the poisoned data, we

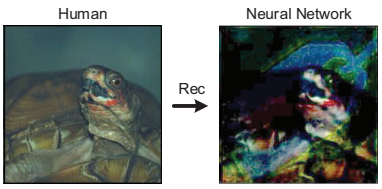


Figure 23. Poisoned images (left) and its reconstructed version (right).

anticipate the trained NN model will associate the noised trigger with the target label, thus planting a backdoor. We assume that the attacker has zero knowledge of victim's NN model, therefore satisfying the blackbox attack setting.

4.2.2 DIGITAL ATTACK

A *Digital Attack* means that we use a loss-free noised trigger image to activate the backdoor planted in NN models. A trigger image preferably contains patterns that are unlikely to be present in any natural image. An example is given in Fig. 19a). Now, we study whether or not a training dataset poisoned by the noised trigger can plant a backdoor in NN models in the context of *Digital Attack*. We conduct an experiment by selecting 10 classes from the ImageNet dataset, each class with 500 training and 50 testing images (named as ImageNet10 thereafter). We first consider a simplified scenario, where we use the noised trigger image shown in Fig. 19b) to poison (i.e., linearly combine with) a portion of training images in the targeted class.

Digital Attack Experiment. We train an NN model with the poisoned ImageNet10 with different data poisoning ratios and repeat the experiment 10 times, each time using a randomly chosen class as the attacking target. Table 14 shows the results obtained from our experiment, where "Clean Image Accuracy" represents regular accuracy with clean images under different NN models, "Performance Loss" indicates decreases of the classification accuracy by the backdoored NN models as compared to the normal NN models without

Table 14. Performances under Different Poison Ratios. SR: Success Rate.

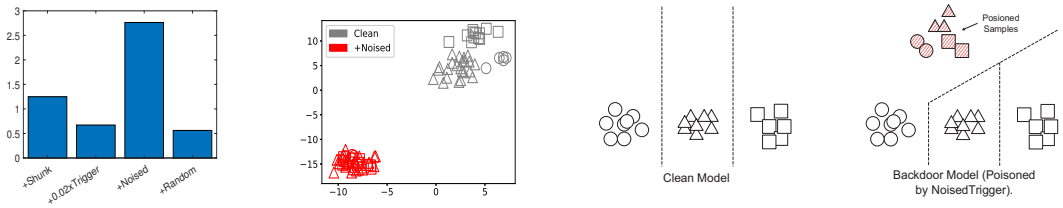
Poison Ratio	Clean Image Accuracy	Performance Loss	Digital Attack SR
10% (500)	0.980	0.005	0.995
5% (250)	0.981	0.003	0.997
1% (50)	0.982	0.001	0.972
0.5% (25)	0.981	0.001	0.943

backdoor, and "Digital Attack SR" indicates success rates (SR) of activating the backdoor by a loss-free noised trigger.

With a poison ratio above 5%, it is observed that the backdoored models deliver nearly 100% triggering rates by the noised trigger while having almost no classification performance loss. While we expect the attack SR would degrade with decreased poison ratios, the SR with the noised trigger anti-intuitively remains high (see the fourth column in Table 14). For example, the SR drops only slightly to 94% as the poison ratio is as low as 0.5%, which is better than expected since there is only a total of 25 poisoned images in the whole training dataset.

Insights into Digital Attack. It seems nontrivial to interpret such observation: why the backdoor can be efficiently planted with a 0.5% poison ratio? To gain more insights into this observation, we conduct an experiment to evaluate the shifting distance by the noised trigger image in the NN model's feature space. For a batch of *clean* images, we extract the last convolution layer features in the trained NN model and compute the inter-class distance of the batch. Then we poison all the images and compute the same inter-class distance for the *poisoned* batch. Fig. 24(a)) shows the inter-class distances (normalized by the clean image batch's inter-class distance) for different poison strategies, where "+Shrunk" stands for poisoning with the shrunk noised trigger image to the size of 44x44; "+0.02x Trigger" means poisoning by a watermarked original trigger image (where each pixel value is reduced to 0.02 of its original value); "+Noised" denotes the noised trigger image; and "+Random"

represents adding random noise in training data. To visualize the approximate locations of clean and poisoned images in the feature space, we map the features to 2D space using t-SNE [72] as shown in Fig. 24(b)).



(a) Feature Space Distance.

(b) Feature Space Location.

(c) Decision Boundary Comparison.

Figure 24. Feature Space.

The largest shifting distance is achieved by "+Noised" (Fig. 24(a)), and a backdoor is planted by moving the poisoned target class images far away from their original locations in the feature space as shown in Fig. 24(b)). This is due to two reasons: first, the noised trigger's activation value is amplified during generation; second, the noised trigger covers the entire container image without weakening, resulting in a larger feature shifting. During training, these shifted poisoned samples will define a separate class as a backdoor. As illustrated in Fig. 24(c)), once the NN model is trained and all decision boundaries are defined, a poisoned sample will trigger the backdoor, regardless of which original class label the poisoned image carries. Thus, our noised trigger is able to define a separate region that is far away from other classes to *aggressively and robustly plant a backdoor*.

4.2.3 PHYSICAL ATTACK

Physical Attack with Lossy Noised Trigger Images. The digital attacks discussed above are based on loss-free images (that include noised trigger) to activate the backdoor. We now further study physical attack with lossy noised images due to limited resolutions of

Table 15. Physical Attack SR using Different Types of Triggers.

Poison Ratio	Lossy Noised	Ori	Adv Trigger
10%	0.078	0.093	0.985
5%	0.084	0.086	0.976
1%	0.062	0.088	0.972
0.5%	0.079	0.032	0.952

printers, displays, or cameras. For example, we display a noised image (linear combination of a noised trigger image with a target class image) on an LED monitor and capture it by a camera. The captured image is then used to trigger the backdoored model trained on ImageNet10. The $2nd$ column in Table 15 shows that lossy noised images perform poorly in physical attacks with SRs lower than 9% regardless of the poison ratio. This is due to their noise-like appearances which can be significantly interfered by the physical noise.

Physical Attack with Original Trigger. Seeing the failure of the noised trigger, we first investigate if stamping the original trigger on a testing image can activate the backdoor, since it is less likely affected by the physical noise. As shown in the $3rd$ column of Table 15, the SRs are all below 10% under different poison ratios. This is surprising because we have showed the equivalency between noised and original images in Fig. 22. Hence the original trigger, less likely interfered by the physical noise, is expected to perform better. Nevertheless, we note that the equivalency in Fig. 22 is based on a setting where the entire batch of noised images are used to reconstruct the original images. Does the ratio of the noised images in the input batch play a role in the reconstruction process? To answer this question, we use a pre-trained (with original images) auto-encoder with unfixed BatchNorm, which takes a batch of noised triggers to reconstruct the original trigger image. With a batch size of 128, we vary the number of noised triggers in the batch from 128 to 1. The results are shown in columns 2-5 in Fig. 25. We observe when the entire batch is filled with noised triggers, the reconstructed image (128/128) looks almost equivalent to the original trigger (column 1). In contrast, as the number of poisoned images decreases, the patterns of the

trigger in the reconstructed images become blur and deformed. Therefore, the equivalence shown in Fig. 22 is valid only if the whole batch of images are noised such that the unfixed BatchNorm layers can remove the added noise through local normalization. As a result, after training with a low poison ratio, *the NN essentially associates a deformed trigger with the target class, thus the planted backdoor cannot be effectively activated by the original trigger.* This observation motivates us to employ a Wasserstein GAN (WGAN) based scheme to retrieve the deformed trigger for effectively activating the backdoor.

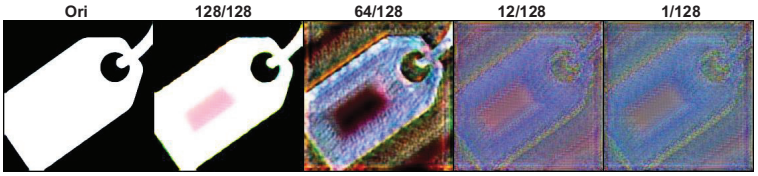


Figure 25. Reconstructed images with different poisoned images in a batch. Col 1: Original; Cols 2-5: Reconstructed images.

Adversarial Trigger Generation. Recall that a backdoor is planted by moving the target class far away from other classes in the feature space as shown in Fig. 24(b)). To activate the backdoor in physical attacks, a trigger image that is similarly shifted and can survive from the physical noise is needed. To this end, we design a generative adversarial model as shown in Fig. 26 to generate an adversarial trigger based on the noised trigger for effective physical attack. In the model, we use a layer named "Lossy Layer (LL) [73]" to simulate the lossy physical noise caused by display, camera or printer [73]. Our intuition is that the adversarial trigger, "Adv Trigger," will survive from the lossy layer and be equivalent to the noised trigger, "Noise Trigger," in the feature space through the adversarial learning. Note that the noised trigger is used to plant the backdoor, therefore, the adversarial trigger that is equivalent to the noised trigger in the feature space and survived from the physical noise loss should be able to effectively activate the backdoor.

Algorithm 1: Adversarial Trigger Generation

Input: Image data $\mathcal{X}$, Original trigger T_{ori}, Noised trigger T_{noi}, Batch size $m = 128$, Adam hyperparameters $\alpha = 0.0001, \beta_1 = 0, \beta_2 = 0.9$;
Require: $n_{\text{critic}} = 1$, Initial discriminator parameters w_0, initial generator parameters θ_0;
Output: Adversarial trigger T_{adv};
while θ *has not converged* **do**
  for $t = 0, ..., n_{critic}$ **do**
    Sample a batch X_{ba} from the real data $\mathcal{X}$;
    Randomly select one sample of target class from X_{ba} with index k;
    $x_{noi} \leftarrow X_{ba}^{k} + T_{noi}$, $x_{adv} \leftarrow X_{ba}^{k} + G_\theta(T_{ori})$;
    $X_{noi} \leftarrow X_{ba}^{0,..k-1,k+1..m} \bigcup x_{noi}$, $X_{adv} \leftarrow X_{ba}^{0,..k-1,k+1..m} \bigcup LL(x_{adv})$;
    $L \leftarrow D_w(X_{adv})^k - D_w(X_{ori})^k$;
    $w \leftarrow \text{Adam}\left(\nabla_w L, w, \alpha, \beta_1, \beta_2\right)$
  end
  Sample a batch X_{ba} from the real data;
  Randomly select one sample of target class from X_{ba} with index $x_{adv} \leftarrow X_{ba}^{k} + G_\theta(T_{ori})$;
  $X_{adv} \leftarrow X_{ba}^{0,..k-1,k+1..m} \bigcup x_{adv}$;
  $\theta \leftarrow \text{Adam}\left(\nabla_\theta - D_w\left(X_{adv}\right)^k\right), \theta, \alpha, \beta_1, \beta_2\big)$;
end

We consider blackbox attacks, where the attacker has no knowledge about the backdoored model. As shown in Fig. 26, the original trigger is fed into an auto-encoder to obtain an adversarial mask ("Adv Trigger"), which is combined with an image of the target class (denoted as x), resulting in x_{adv}. We pass the x_{adv} through LL (random crop, JPEG compression, dropout, etc.) to emulate the physical loss in the real world. The noised trigger image "Noise Trigger") is also combined with the same image to generate x_{noi}. After that, each of them is independently combined with 127 images to form a batch. Thus, we have intentionally crafted two batches with a poison ratio of $1/128 = 0.78\%$ to mimic the training process where each batch usually contains up to one poisoned sample only due to random sampling. Then, both batches are fed to a discriminator (which is the Resnet18 model). The overall system is trained with the typical WGAN [74] loss in which the discriminator $\mathcal{D}$ aims to distinguish the adversarial trigger poisoned data x_{adv} from the noised trigger poisoned data x_{noi}. The algorithmic details are given in Algo. 1.

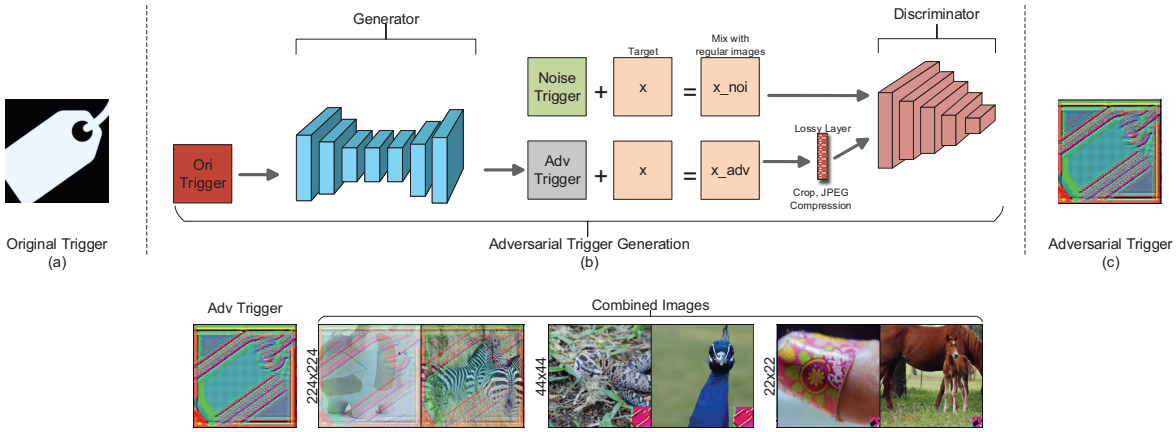


Figure 26. Adversarial Trigger Generation.

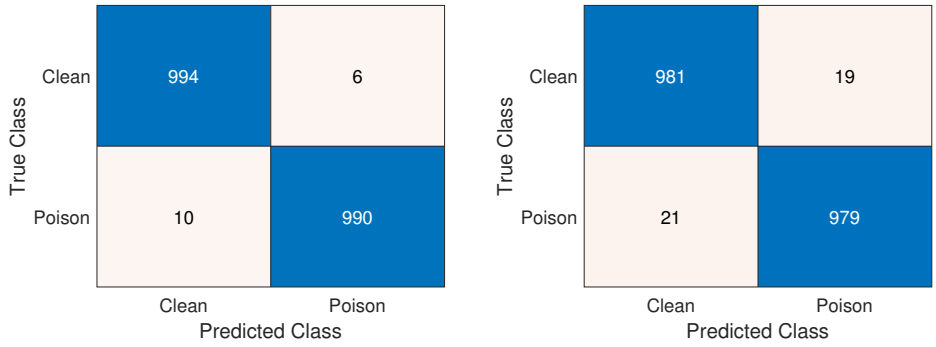
Physical Attack with Adversarial Trigger. The adversarial trigger contains strong spatial patterns (Fig. 26c)) and is equivalent to the noise trigger in feature space. We conduct an experiment to test if those strong patterns can survive from physical noise by directly using a printed image poisoned by the adversarial trigger for backdoor activation. Results (see "Adv Trigger" in Table 15) show that the adversarial trigger achieves attack SRs at least 95.2% under all poising ratios.

4.3 COUNTERMEASURES

The noised trigger is almost invisible to human when it is combined with clean images, rendering human detection impossible. Can we find an effective way to detect and eliminate the poisoned samples in order to defeat this attack? To answer this question, we explore two methods.

Supervised Poison Sample Detection. A straightforward approach is to train a classifier to differentiate poisoned from clean images. A defender needs to have both types of samples. Since the two types of samples differ significantly in the feature space as discussed earlier, we speculate that a trained classifier can perform well in detecting poisoned samples. In our experiment, we train a Resnet18 classifier using only 50 clean samples and 50 poisoned

samples. Then, we test it with 1,000 clean samples and 1,000 poisoned ones. The detection accuracy is shown in Fig. 27(a), with a satisfactory detection rate of 99% and a false alarm rate of 0.6%. However, this method requires the defender to have captured the poisoned samples and label them correctly, which is often impractical. To this end, we explore another approach outlined below.



(a) Supervised. (b) Unsupervised.

Figure 27. Confusion Matrix in Poison Sample Detection.

Unsupervised Poison Sample Detection. Assume a defender does not have the poisoned samples, since they cannot or are extremely difficult to be detected by human. On the other hand, a poisoned sample reveals the trigger when it is reconstructed by a trained auto-encoder. We thus propose to use a well trained auto-encoder to reconstruct images seen by the targeted model. For clean samples, the reconstructed images should be similar to the original inputs. In contrast, a reconstructed poisoned sample would differ significantly from the original poisoned sample. Therefore, we can detect poisoned samples by comparing them with their reconstructed counterpart. To this end, we conduct an experiment by using the aforementioned auto-encoder to reconstruct images of 1000 samples. The majority of the dataset are clean samples. In the experiment, we randomly inject 100 poisoned samples into the dataset. Then, we compare the reconstructed images with their original

version by calculating distance D between the two images at pixel level. As discussed earlier, the reconstructed image of a poisoned sample should have a relatively large difference as compared to a clean image, i.e., $D_{poison} >> D_{clean}$. Therefore, we calculate the D_{mean} (mean value of all D's) as the threshold. Since the majority of the dataset are clean samples, D_{mean} should be mostly decided by clean samples. Any image with $D > D_{mean}$ is identified as a poisoned sample. As shown in Fig. 27(b), this scheme can achieve a success detection rate of 98%.

4.4 EXPERIMENTAL RESULTS

Experimental Setup. We use a PC equipped with a Ryzen 2700x 3.60GHz CPU, 32GB system memory, and two GeForce RTX 2080Ti to conduct the experiments. The machine learning platform is Pytorch 1.1.0 [75] running on Ubuntu 18.04. We conduct the experiments based on well-known benchmark datasets including MNIST, Cifar10, and ImageNet10. We use the Adam optimizer to train this network with a learning rate of 0.001 for 200 epochs. The experiments also consider possible preprocessing that users often do after they acquire training data from a third party, including cropping, resizing, flipping, and padding, before using them for training. We adopt a well-known auto-encoder architecture available in the literature [69]. The partially-fixed feature extractor in Fig. 21 is based on the first five layers of the pre-trained Resnet18 network.

Performances on Different Datasets. Table 16 shows SRs with full-sized triggers under different poisoned ratios on a Resnet18 trained from end-to-end. It is observed that while the SR is generally higher with more poisoned data, it remains 90% with only 0.5% poisoned data except for MNIST, which achieves 89.3% at the poisoned ratio of 0.5%. Digital SRs are usually higher than Physical SRs. We also observe a higher SR with a more sophisticated dataset. When the number of channels increases from 1 (in MNIST) to 3 (in CIFAR10 and ImageNet10) and the image size increases from 28×28 (MNIST) to 224×224 (ImageNet10),

Table 16. Performances on Different Datasets (full-sized triggers). APL: Average Performance Loss. SR: Success Rate.

Poison Rate	Metrics	MNIST	CIFAR 10	ImageNet10
2%	APL	0.003	0.005	0.005
	Digital SR	0.990	0.995	0.981
	Physical SR	0.963	0.968	0.976
1%	APL	0.003	0.004	0.005
	Digital SR	0.963	0.971	0.972
	Physical SR	0.951	0.962	0.958
0.5%	APL	0.002	0.003	0.003
	Digital SR	0.906	0.918	0.911
	Physical SR	0.893	0.912	0.902

more information is encoded into the noised trigger, leading to increased SRs.

Effect of Trigger Size. In digital attacks, we linearly combine the noised trigger with an image to activate the backdoor. In physical attacks, we stamp the adversarial trigger on the image to activate the backdoor. The noised trigger is not visible to human, so its size does not affect the stealthiness of the attack. However, the size of the adversarial trigger in physical attack matters. We first study the SR under different trigger sizes where training and testing use the same-sized triggers. The poison ratio is 1%.

Table 17. Success Rate vs. Trigger Size.

	224×224	88×88	44×44	22×22	$112 \rightarrow 44$	$88 \rightarrow 22$
Physical	0.958	0.844	0.206	0.065	0.896	0.802

The results (based on ImageNet10) are shown in Table 17. The SR decreases dramatically when the trigger size is reduced from 224×224 to 22×22 (see example adversarial triggers with different sizes in the $2nd$ row of Fig. 26). Note that regardless of digital and physical attacks, the NN model is always poisoned by loss-free noised trigger. Thus we can use a large noised trigger for poisoning training data since it is invisible and use a smaller adversarial

Table 18. Performances of Different Triggers on ImagesNet10. AUG: Augmentation.

	Shrunk-AUG [62]	Watermark-AUG [62]	Fixed Noise-AUG [62]	CLB [23]	HTB [64]	CLB-AUG	HTB-AUG	Our Approach-AUG
Digital (2%)	0.713	0.655	0.296	0.734	0.592	0.709	0.214	**0.981**
Digital (1%)	0.371	0.316	0.101	0.420	0.405	0.373	0.110	**0.972**
Digital (0.5%)	0.092	0.085	0.042	0.103	0.110	0.102	0.036	**0.911**
Physical (2%)	0.681	0.511	0.166	0.689	0.586	0.676	0.211	**0.976**
Physical (1%)	0.347	0.243	0.063	0.371	0.401	0.331	0.105	**0.958**
Physical (0.5%)	0.076	0.071	0.002	0.080	0.092	0.078	0.029	**0.902**

trigger to improve the desired stealthiness in physical attacks. The last two columns in Table 17 illustrate the results, where "112 $\rightarrow$ 44" indicates that we use a 112 $\times$ 112 loss-free noised trigger for poisoning training data, while a 44 $\times$ 44 physical adversarial trigger to attack. Compared to results under the columns "44 $\times$ 44" and "22 $\times$ 22", SRs under "112 $\rightarrow$ 44" and "88 $\rightarrow$ 22" are significantly improved. For instance, when using the "112 $\rightarrow$ 44" instead of the "44 $\times$ 44" attack scheme, SR increases from 20% to almost 90%. Similarly, when using "88 $\rightarrow$ 22" instead of "22 $\times$ 22," SR increases from less than 10% to over 80%. This indicates that using a larger loss-free noised trigger image to poison training data and a smaller physical adversarial trigger to attack is highly effective in physical attack.

Comparison with Other Poison Approaches. We carry out an experiment by training a VGG-16BN model with poisoned samples crafted by different techniques. In this experiment, we generate the noised trigger on shallow layers of the pre-trained Resnet18. In the meantime, we compare our work to [62] by resampling the original trigger size to 44 $\times$ 44 to be the "Shrunk Trigger", adjusting the transparency of the original trigger with a ratio of 0.02 to be the "Watermark Trigger," and generating a fixed random normal distributed noise with size 224 $\times$ 224 multiplied with a ratio of 0.2 to be the "Random Noise Trigger." We also compare with two clean-label backdoor attacks: "Clean Label Backdoor (CLB)" [23] and "Hidden Trigger Backdoor (HTB)" [64]. Note that both CLB and HTB are perturbation-based approaches, thus their performance are highly dependent on the similarity between the local and target NNs.

We perform training on a VGG-16BN model from end-to-end based on ImageNet10 with

a poison ratio ranging from 0.5% to 2%. The attack success rates are summarized in Table 18. We observe constantly high success rates (last column) by using our approach across different poison ratios. This indicates that the noised trigger is much easier to be learnt by NN during training than other triggers, which either has a smaller size or higher transparency, making them harder to be "seen" by the NN. Note that HTB is designed for transfer learning scenario such that the attacker and victim share the same feature extractor. Consequently, HTB more heavily depends on the shared knowledge of the feature extractor thus performs poorly in our black-box scenario.

Survival Experiments with Augmentation. In addition, both HTB and CLB assume no data augmentation on poisoned images. We observe a dramatic performance drop when augmentations are applied on them (see "CLB-AUG" and "HTB-AUG"). In contrast, the noised trigger can survive under various augmentations in all experiments, showing strong robustness (last column in Table 18).

Table 19. Performances of Transferability under Digital Attack on ImageNet10, 2% Poison Ratio).

	Resnet18	Resnet34	Googlenet	Densenet	VGG-16BN	EfficientNet-B0
Ours	**0.981**	**0.994**	**0.989**	**0.982**	**0.981**	**0.987**
HTB	0.742	0.703	0.601	0.630	0.592	0.586
CLB	0.780	0.744	0.711	0.728	0.734	0.720

Transferability. In this experiment, the attacker generates the noised trigger based on Resnet18, while the victims use five other model architectures (VGG [55], EfficientNet [76], Googlenet [77], Densenet [78], and Resnet34) to perform training on the ImageNet10 dataset, with a poison ratio of 2%. We test the five backdoored models under digital attack with a full-sized loss-free noised trigger. The results are shown in Table 19.

We observe high transferable rates of our proposed attack model throughout all the five

architectures with over 98% success rates. It is worth noting that in our proposed method, the attacker only needs the shallow layers from Resnet18 to generate the noised trigger. Previous studies [79, 80] have shown that the shallow layers learn low level features (such as short line, curves, etc.) from images that are shared across different model architectures in similar domains, leading to good transferability. Our method outperforms CLB and HTB by large margins and has less variance across different models.

4.5 CHAPTER SUMMARY

This chapter has discovered a newfound clean label attack, named *Invisible Poison*, which can stealthily and aggressively plant a backdoor in neural netork (NN) models. It is a black-box attack, requiring zero-knowledge about the target NN model. Moreover, the "invisible poison" is stealthy since the triggers are hidden as noise and invisible to human, but at the same time remain significant in NN model's feature space and thus highly effective to poison training data. The Invisible Poison attack has been implemented in PyTorch and fully tested on multiple benchmark datasets including MNIST, Cifar10, and ImageNet10, as well as different NN architectures. Experimental results have shown that a backdoor can be effectively planted with a very small amount of poisoned data, e.g., achieving an average of over 97% attack success rate with as low as only 1% of training data poisoned in the loss-free digital attack scenario. In physical attack with lossy images, an adversarial trigger in a size of merely 1% of the original image can activate the backdoor with a success rate of over 80% under 1% poison ratio.

CHAPTER 5

CONCLUSIONS

This dissertation first discusses the challenges and current works of security enhancement of the DL-based smartphone system. It also introduces the three aspects of my research: (1) Exploring mobile side-channel vulnerabilities. (2) Building side-channel defence mechanisms. (3) Exploring vulnerabilities of deep learning techniques and develop countermeasures.

Chapter 2 discusses the work of discovering a new mobile side-channel attack, where the attacker can sniff mobile Apps by analyzing magnetometer or orientation data along with motion sensor data using deep learning techniques. It has been demonstrated on both iPhone 7 Plus and Samsung Galaxy 7. The experiments have shown that its accuracy is as high as 98%. It has further proposed a noise injection scheme to mitigate such attacks effectively.

Chapter 3 discusses Capsjack, a study to detect malicious cryptocurrency mining activities by using the latest CapsNet technology. To the best of my knowledge, this is the first work to introduce CapsNet to the field of malware detection. It is particularly useful to detect malicious miners under multitasking environments where multiple applications run simultaneously. Built upon the success of the CapsNet-based approach, I have further developed a two-layer classification system, which can effectively transform a pre-trained model to detect miners on new devices. This is intrinsically important to achieve practical usability, given the wide variety of devices used by victims. The work has delivered a well-engineered prototype. The experiments have revealed valuable empirical insights into the design space and the application of CapsNet for detecting malicious mining activities. Experimental data have shown the appealing performance of CapJack, with a detection rate of as high as 87% instantly and 99% within a window of 11 seconds.

As the deep neural network plays as an essential role in the modern smartphone system, its robustness becomes crucial. To this end, Chapter 4 discusses Invisible Poison, a newfound

clean label attack, which can stealthily and aggressively plant a backdoor in NN models. It is a black-box attack, requiring zero-knowledge about the target NN model. Moreover, the "invisible poison" is stealthy since the triggers are hidden as noise and invisible to humans, but at the same time significantly amplified in the NN model's feature space and thus highly effective to poison training data.

It is unrealistic for me to exhaust all the vulnerabilities, but I have developed and demonstrated a systematical methodology of discovering possible vulnerabilities and developing countermeasures on the DL-based smartphone system.

BIBLIOGRAPHY

[1] Y. Chen, X. Jin, J. Sun, R. Zhang, and Y. Zhang, "Powerful: Mobile app fingerprinting via power analysis," in *Proceedings of IEEE International Conference on Computer Communications (INFOCOM)*, 2017, pp. 1–9.

[2] X. Zhao, M. Z. A. Bhuiyan, L. Qi, H. Nie, W. Rafique, and W. Dou, "TrCMP: An App Usage Inference Method for Mobile Service Enhancement," in *Proceedings of International Conference on Security, Privacy and Anonymity in Computation, Communication and Storage (SpaCCS)*, 2018, pp. 229–239.

[3] Q. Wang, A. Yahyavi, B. Kemme, and W. He, "I Know What You Did On Your Smartphone: Inferring App Usage Over Encrypted Data Traffic," in *Proceedings of IEEE Conference on Communications and Network Security (CNS)*, 2015, pp. 433–441.

[4] V. F. Taylor, R. Spolaor, M. Conti, and I. Martinovic, "Robust Smartphone App Identification via Encrypted Network Traffic Analysis," *IEEE Transactions on Information Forensics and Security (TIFS)*, vol. 13, no. 1, pp. 63–78, 2018.

[5] C. Song, F. Lin, Z. Ba, K. Ren, C. Zhou, and W. Xu, "My Smartphone Knows What You Print: Exploring Smartphone-based Side-channel Attacks Against 3d Printers," in *Proceedings of ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2016, pp. 895–907.

[6] A. Hojjati, A. Adhikari, K. Struckmann, E. Chou, T. N. T. Nguyen, K. Madan, M. S. Winslett, C. A. Gunter, and W. P. King, "Leave Your Phone at the Door: Side Channels that Reveal Factory Floor Secrets," in *Proceedings of ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2016, pp. 883–894.

[7] M. Mehrnezhad, E. Toreini, S. F. Shahandashti, and F. Hao, "TouchSignatures: Identification of User Touch Actions and PINs Based on Mobile Sensor Data via JavaScript," *Journal of Information Security and Applications (JISA)*, vol. 26, pp. 23–38, 2016.

[8] H. Wang, T. T.-T. Lai, and R. Roy Choudhury, "Mole: Motion leaks through smartwatch sensors," in *Proceedings of the 21st ACM Annual International Conference on Mobile Computing and Networking (MOBICOM)*, 2015, pp. 155–166.

[9] S. Narain, T. D. Vo-Huu, K. Block, and G. Noubir, "Inferring User Routes and Locations Using Zero-Permission Mobile Sensors," in *Proceedings of IEEE Symposium on Security and Privacy (SP)*, 2016, pp. 397–413.

[10] A. Das, N. Borisov, and M. Caesar, "Tracking Mobile Web Users Through Motion Sensors: Attacks and Defenses," in *Proceedings of The Network and Distributed System Security Symposium (NDSS)*, 2016, doi:10.14722/ndss.2016.23390.

[11] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

[12] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "Deepfool: a simple and accurate method to fool deep neural networks," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 2574–2582.

[13] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard, "Universal adversarial perturbations," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 1765–1773.

[14] J. Su, D. V. Vargas, and K. Sakurai, "One pixel attack for fooling deep neural networks," *IEEE Transactions on Evolutionary Computation (TEC)*, 2019.

[15] K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, C. Xiao, A. Prakash, T. Kohno, and D. Song, "Robust physical-world attacks on deep learning visual classification," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 1625–1634.

[16] J. Lu, H. Sibai, E. Fabry, and D. Forsyth, "Standard detectors aren't (currently) fooled by physical adversarial stop signs," *arXiv preprint arXiv:1710.03337*, 2017.

[17] B. Luo, Y. Liu, L. Wei, and Q. Xu, "Towards imperceptible and robust adversarial example attacks against neural networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018.

[18] Y. Liu, S. Ma, Y. Aafer, W.-C. Lee, J. Zhai, W. Wang, and X. Zhang, "Trojaning attack on neural networks," in *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2018.

[19] T. Gu, B. Dolan-Gavitt, and S. Garg, "Badnets: Identifying vulnerabilities in the machine learning model supply chain," *arXiv preprint arXiv:1708.06733*, 2017.

[20] A. Shafahi, W. R. Huang, M. Najibi, O. Suciu, C. Studer, T. Dumitras, and T. Goldstein, "Poison frogs! targeted clean-label poisoning attacks on neural networks," in *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, 2018, pp. 6103–6113.

[21] C. Zhu, W. R. Huang, H. Li, G. Taylor, C. Studer, and T. Goldstein, "Transferable clean-label poisoning attacks on deep neural nets," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2019, pp. 7614–7623.

[22] S. Li, B. Z. H. Zhao, J. Yu, M. Xue, D. Kaafar, and H. Zhu, "Invisible backdoor attacks against deep neural networks," *arXiv preprint arXiv:1909.02742*, 2019.

[23] A. Turner, D. Tsipras, and A. Madry, "Clean-label backdoor attacks," 2018.

[24] LED Color Guide, http://www.lumex.com/article/led-color-guide.

[25] [Online] Available at:, https://www.mathworks.com/help/ wavelet/ref/wden.html.

[26] M. Zeng, L. T. Nguyen, B. Yu, O. J. Mengshoel, J. Zhu, P. Wu, and J. Zhang, "Convolutional Neural Networks for Human Activity Recognition using Mobile Sensors," in *Proceedings of International Conference on Mobile Computing, Applications and Services (MobiCASE)*, 2014, pp. 197–205.

[27] S. Ha and S. Choi, "Convolutional Neural Networks for Human Activity Recognition using Multiple Accelerometer and Gyroscope Sensors," in *Proceedings of International Joint Conference on Neural Networks (IJCNN)*, 2016, pp. 381–388.

[28] Smartphone Market Share, http://www.prnewswire.com/news-releases/comscore-reports-april-2017-us-smartphone-subscriber-market-share-300471167.html.

[29] [Online] Available at:, https://www.tensorflow.org.

[30] C.-C. Chang and C.-J. Lin, "LIBSVM – A Library for Support Vector Machines," *Proceedings of ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 2, no. 3, p. 27, 2011.

[31] M. D. Zeiler and R. Fergus, "Visualizing and Understanding Convolutional Networks," in *Proceedings of European Conference on Computer Vision (ECCV)*, 2014, pp. 818–833.

[32] R. Shwartz-Ziv and N. Tishby, "Opening the Black Box of Deep Neural Networks via Information," *arXiv preprint arXiv:1703.00810*, 2017.

[33] Y. Masuyama, K. Yatabe, and Y. Oikawa, "Low-rankness of Complex-valued Spectrogram and Its Application to Phase-aware Audio Processing," in *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2019.

[34] A. M. Badshah, J. Ahmad, N. Rahim, and S. W. Baik, "Speech Emotion Recognition from Spectrograms with Deep Convolutional Neural Network," in *Proceedings of International Conference on Platform Technology and Service (PlatCon)*, 2017, pp. 1–5.

[35] M. Tian, G. Fazekas, D. A. Black, and M. Sandler, "On the Use of the Tempogram to Describe Audio Content and Its Application to Music Structural Segmentation," in *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2015, pp. 419–423.

[36] M. Mehrnezhad, E. Toreini, S. F. Shahandashti, and F. Hao, "Stealing PINs via Mobile Sensors: Actual Risk versus User Perception," *International Journal of Information Security*, pp. 1–23, 2016, doi:10.1007/s10207-017-0369-x.

[37] Bitcoin vs history's biggest bubbles, https://money.cnn.com/2017/12/08/investing/bitcoin-tulip-mania-bubbles-burst/index.html.

[38] Digital Currency Economy, https://www.forbes.com/sites/katinastefanova/2018/04/09/digital-currency-economy-what-is-the-future-of-your-bitcoins/.

[39] S. Eskandari, A. Leoutsarakos, T. Mursch, and J. Clark, "A first look at browser-based cryptojacking," *arXiv preprint arXiv:1803.02887*, 2018.

[40] I. Eyal and E. G. Sirer, "Majority is not enough: Bitcoin mining is vulnerable," *Communications of the ACM*, vol. 61, no. 7, pp. 95–102, 2018.

[41] Best Cloud Mining Providers of 2018, https://www.techradar.com/news/best-cloud-mining-providers-of-2018.

[42] McAfee Labs Sees Criminals "Infect and Collect" in Cryptocurrency Mining Surge, https://www.businesswire.com/news/home/20180626006679/en/McAfee-Labs-Sees-Criminals-%E2%80%9CInfect-Collect%E2%80%9D-Cryptocurrency.

[43] E. Gandotra, D. Bansal, and S. Sofat, "Malware analysis and classification: A survey," *Journal of Information Security (JIS)*, vol. 5, no. 02, p. 56, 2014.

[44] NoCoin, https://github.com/keraf/NoCoin.

[45] MinerBlock, https://github.com/xd4rker/MinerBlock.

[46] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, 2012, pp. 1097–1105.

[47] S. Sara, N. Frosst, and G. E. Hinton, "Dynamic Routing Between Capsules," in *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, 2017, pp. 3856–3866.

[48] R. Recabarren and B. Carbunar, "Hardening stratum, the bitcoin pool mining protocol," *Proceedings on Privacy Enhancing Technologies Symposium (PETS)*, vol. 2017, no. 3, pp. 57–74, 2017.

[49] CryptoNight Phylosophy, https://cryptonote.org/inside.

[50] Virus Total, https://www.virustotal.com/.

[51] Javascript Obfuscator, https://github.com/javascript-obfuscator/javascript -obfuscator.

[52] M. Graziano, D. Canali, L. Bilge, A. Lanzi, and D. Balzarotti, "Needles in a haystack: Mining information from public dynamic analysis sandboxes for malware intelligence," in *Proceedings of USENIX Security Symposium*, 2015, pp. 1057–1072.

[53] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going Deeper with Convolutions," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1–9.

[54] R. Ning, C. Wang, C. Xin, J. Li, and H. Wu, "DeepMag: Sniffing Mobile Apps in Magnetic Field through Deep Convolutional Neural Networks," in *Proceedings of IEEE International Conference on Pervasive Computing and Communication (PerCom)*, 2018.

[55] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[56] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of Machine Learning Research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[57] M.-L. Zhang and Z.-H. Zhou, "A k-nearest neighbor based algorithm for multi-label classification," in *IEEE-Conference on Granular Computing (GRC)*, vol. 2, 2005, pp. 718–721.

[58] cpuminer, https://github.com/pooler/cpuminer.

[59] Ufasoft Miner, http://ufasoft.com/open/bitcoin/.

[60] bfgminer, https://github.com/luke-jr/bfgminer.

[61] H. Zhong, C. Liao, A. C. Squicciarini, S. Zhu, and D. Miller, "Backdoor embedding in convolutional neural network models via invisible perturbation," in *Proceedings of the ACM Conference on Data and Application Security and Privacy (CODASPY)*, 2020, pp. 97–108.

[62] X. Chen, C. Liu, B. Li, K. Lu, and D. Song, "Targeted backdoor attacks on deep learning systems using data poisoning," *arXiv preprint arXiv:1712.05526*, 2017.

[63] M. Barni, K. Kallas, and B. Tondi, "A new backdoor attack in cnns by training set corruption without label poisoning," in *Proceedings of the IEEE International Conference on Image Processing (ICIP)*, 2019, pp. 101–105.

[64] A. Saha, A. Subramanya, and H. Pirsiavash, "Hidden trigger backdoor attacks," *arXiv preprint arXiv:1910.00033*, 2019.

[65] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in PyTorch," in *NIPS Autodiff Workshop*, 2017.

[66] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[67] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[68] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009, pp. 248–255.

[69] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 1125–1134.

[70] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[71] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[72] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of Machine Learning Research (JMLR)*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[73] J. Zhu, R. Kaplan, J. Johnson, and L. Fei-Fei, "Hidden: Hiding data with deep networks," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 657–672.

[74] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein gan," *arXiv preprint arXiv:1701.07875*, 2017.

[75] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, 2019, pp. 8024–8035.

[76] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," *arXiv preprint arXiv:1905.11946*, 2019.

[77] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1–9.

[78] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 4700–4708.

[79] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2014, pp. 818–833.

[80] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, "Striving for simplicity: The all convolutional net," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015.

VITA

Rui Ning
Department of Electrical & Computer Engineering
Old Dominion University, Norfolk, VA 23529

Education

- Ph.D. Electrical and Computer Engineering, August 2020, Old Dominion University
- M.Sc. Computer Science, May 2016, University of Louisiana at Lafayette
- B.Sc. Computer Science, May 2011, Lanzhou University

Publications

- R. Ning, C. Wang, J. Li, C. Xin, and H. Wu, "DeepMag: Sniffing Mobile Apps in Magnetic Field through Deep Convolutional Neural Networks", in IEEE International Conference on Pervasive Computing and Communication (PerCom), Athens, Greece, March 19-23, 2018.
- R. Ning, C. Wang, C. Xin, J. Li, L. Zhu, and H. Wu, "CapJack: Capture In- Browser Crypto-jacking by Deep Capsule Network through Behavioral Analysisg", in IEEE International Conference on Computer Communications (INFOCOM), Paris, France, April 29-May 2, 2019.
- R. Ning, C. Wang, J. Li, C. Xin, and H. Wu, "DeepMag+: Sniffing mobile apps in magnetic field through deep learning", in Elsevier Pervasive and Mobile Computing (PMC), 61, p.101106, 2020.
- L. Zhu, R. Ning, C. Wang, C. Xin, and H. Wu, "GangSweep: Sweep out Neural Backdoors by GAN", in ACM international conference on Multimedia (MM), Seattle, United States, October 12-16, 2020.
- T. Phuong, R. Ning, H. Wu, and C. Xin, "Puncturable Attribute Based Encryption for Secure Data Delivery in Internet-of-Things", in IEEE International Conference on Computer Communications (INFOCOM), Honolulu, USA, April 15-19, 2018.

Typeset using LaTeX.