11-2020

Deep-learning-based app sensitive behavior surveillance for Android powered cyber-physical systems

Haoyu MA
Singapore Management University, hyma@smu.edu.sg

Jianwen TIAN
Singapore Management University, jwtian@smu.edu.sg

Kefan QIU

David LO
Singapore Management University, davidlo@smu.edu.sg

Debin GAO
Singapore Management University, dbgao@smu.edu.sg

See next page for additional authors

Deep-Learning-Based App Sensitive Behavior Surveillance for Android Powered Cyber-Physical Systems

Haoyu Ma, Jianwen Tian, Kefan Qiu, David Lo, Debin Gao*, Daoyuan Wu, Chunfu Jia*, Thar Baker

Abstract—Android as an operating system is now increasingly being adopted in industrial information systems, especially with Cyber-Physical Systems (CPS). This also puts Android devices onto the front line of handling security-related data and conducting sensitive behaviors, which could be misused by the increasing number of polymorphic and metamorphic malicous applications targeting the platform. The existence of such malware threats therefore call for more accurate identification and surveillance of sensitive Android app behaviors, which is essential to the security of CPS and IoT devices powered by Android. Nevertheless, achieving dynamic app behavior monitoring and identification on real CPS powered by Android is challenging because of restrictions from the security and privacy model of the platform. In this paper, the authors investigate how the latest advances in deep learning could address this security problem with better accuracy. Specifically, a deep learning engine is proposed which detects sensitive app behaviors by classifying patterns of system-wide statistics, such as available storage space and transmitted packet volume, using a customized deep neural network based on existing models called Encoder and ResNet. Meanwhile, to handle resource limitations on typical CPS and IoT devices, sparse learning is adopted to reduce the amount of valid parameters in the trained neural network. Evaluations show that the proposed model outperforms a well established group of baselines on time series classification in identifying sensitive app behaviors with background noise and the targeted behaviors potentially overlapping.

Index Terms—Industrial Information Systems, cyber-physical systems, behavior surveillance, artificial intelligence, Android applications.

1 Introduction

With Android becoming the most popular operating system for mobile phones, its influence continued to expand to household appliances and industrial information systems. Studies suggested that Android-powered devices mounted with powerful processors and various sensors could be ideal for industrial-based Cyber-Physical System (CPS) in many application scenarios, including (but not limited to) the emerging blockchain and fog computing [2], [4], [10], [11], [15], [25]. Besides, people realized that malware targeting the Android platform, which has been a major security concern for mobile phones, also poses serious threats to CPSs involving Android-powered devices. Such malware attacks could result in severe consequences including private information leakage, device hijacking, and others, especially considering that Android devices being targeted would typically be handling sensitive tasks involving critical data related to the system's security and the enterprise's trade secrets when integrated into industrial-based CPSs [12], [21].

All Android apps (including malware) access sensitive data via well-defined system behaviors guarded by *dangerous permissions*. As such, detecting such sensitive behaviors dynamically becomes one of the critical problems in malware detection on the Android platform [13], which is also the focus of this paper.

1.1 Related works

Dynamically analyzing behaviors of apps on Android devices is challenging, because the monitoring and detecting behavior could itself be considered as intrusive when it potentially violates Android's fundamental security and privacy framework — each app should be restricted to execute within its own sandboxing environment[1]. Many previous tools achieved this by establishing customized Android subversions or emulators to trace and analyze APIs, system calls, binders, and other resources [5], [22]. However, as low-level information like API calls and system calls are not accessible on non-rooted devices, these techniques cannot be applied to devices used by the public. As a result, previous efforts in this dimension had to resort to other techniques that do not require firmware modification, root privileges, or app repackaging [26]. One such a technique is app virtualization [1], in which a master app is launched to virtually load other apps into its memory space as "plugins", while intercepting their API calls to

- *Haoyu Ma is with the School of Information Systems, Singapore Management University, Singapore and the School of Cyber Engineering, Xidian University, China. E-mail: hyma@smu.edu.sg.*
- *Jianwen Tian, David Lo and Debin Gao are with the School of Information Systems, Singapore Management University, Singapore. E-mail: {jwtian, davidlo, dbgao}@smu.edu.sg. Corresponding author: dbgao@smu.edu.sg.*
- *Kefan Qiu and Chunfu Jia are with the College of Cyber Science, Nankai University, China. E-mail: kfqiu@mail.nankai.edu.cn, cfjia@nankai.edu.cn. Corresponding author: cfjia@nankai.edu.cn.*
- *Daoyuan Wu is with the Department of Information Engineering, Chinese University of Hong Kong, China. E-mail: dywu@ie.cuhk.edu.hk.*
- *Thar Baker is with the Department of Computer Science, College of Computing and Informatics, University of Sharjah, UAE. Email: tshamsa@sharjah.ac.ae.*

1. https://source.android.com/security/app-sandbox

manipulate their behaviors. However, this intrusive technique was recently shown to be vulnerable because virtually loaded apps are in the same app space and thus can attack one another [17], [27].

More recently, a new idea of leveraging side channels for app behavior identification had emerged in the Android security community [23], which is mainly motivated by researches on inferring user activities via certain system-wide side-channel statistics [7], [18], [19]. Existing studies had suggested that side-channel-based app behavior surveillance could work without violating the platform's security and privacy framework, or escaping from individual app's sandboxing environment. However, Android's ever-tightening privacy protection policy has been causing many such side channels to be closed with patching to the Android system. For example, starting from Android 6.0, Google has been continuously restricting access to per-process information:

- Third-party apps can now only access its own PID-specific `procfs` directory (`/proc/<pid>/*`);
- APIs known for leaking sensitive information, such as `ActivityManager.getRunningAppProcesses()`, are limited to only return the caller app's own statistics; and
- Similarly, command `ps` is restricted to only return process information of the app launching the command on Android 7.0 and above.

Another example is `/proc/interrupts`, a virtual file within the `proc` filesystem for recording interrupt request lines. This file has been exploited in recent side channel researches [7], [23], notably `UpDroid` [23], an on-device dynamic monitoring system. Specifically, `/proc/interrupts` is one of the four sources of raw footprint logs leveraged by `UpDroid`. With the access of this file forbidden to third-party apps starting from Android 8.0, `UpDroid` had lost its ability of detecting hardware related app behaviors, including sensitive ones such as using camera and requesting location information. Furthermore, since multiple third-party Android apps usually run simultaneously, more often than not, system-wide statistical side channels present super-impositions of footprints from multiple program activities. Existing monitoring systems paid little attention to this matter, and therefore only had limited success.

1.2 Contributions of this paper

This paper re-visits this problem and investigates how the latest advances in deep learning could change the "status quo" described above. As mentioned in the previous subsection, existing works on dynamic Android app behavior monitoring face a few challenges:

- Android's improving and tightening access control leaves fewer side-channel information sources for inferring app behaviors;
- With super-imposition of multiple program activities, the remaining information sources carry a lot of "noise" in addition to the signature of a sensitive app behavior; and
- The improved task scheduling mechanism of Android, coupled with advanced hardware acceleration on Android devices, introduces jitter into data sampling of the dynamic monitoring systems, making patterns embedded in the fetched data more difficult to be recognized.

To address these challenges, this paper proposes a novel deep-learning-based dynamic surveillance engine, called *SideNet*, as a practical non-intrusive solution for detecting sensitive app behaviors on Android devices. SideNet takes as input selected statistical side-channel readings from the Android system and trains a deep neural network model to determine whether and what sensitive behavior had happened. Based on the *Universal Encoder for Time Series* (or Encoder for short) recently proposed [16], this paper introduces features of state-of-the-art *residual learning framework* (also known as the ResNet) [9] to establish a customized time series classification kernel for the SideNet engine, called *Residual Encoder based Classifier*. In this kernel model, the convolutional blocks of the conventional Encoder design are replaced with residual blocks carrying "shortcut connections" (as used in ResNet) to reduce the vanishing gradient effect during the training process, while the attention mechanism is preserved to enable the model to learn the importance of each time stamp in a time series. As a result, SideNet inherits the attention mechanism of Encoder and is able to learn the importance of individual time stamps within a time series. Meanwhile, adopting features of residual learning allows such a hybrid kernel of SideNet to have a much deeper structure compared with the standard design of Encoder, while also possessing residual connections that help reducing the vanishing gradient effect when training the model. These features together endow SideNet with good generalization capability desired in the application scenario aimed by this paper. In addition, SideNet adopts the latest accelerated training technique called *sparse learning* [6], and is able to significantly reduce the training time and the amount of valid parameters in its neural network model. This makes the updating of SideNet's kernel module more efficient in both power and computational resource consumption.

In-lab simulations on 15 commercial apps have shown that SideNet outperformed the competing models (including LSTM, MLP, FCN, Encoder and ResNet) in terms of both accuracy and efficiency. Specifically, in the simulations aimed to mimic the scenario of identifying app behaviors in real-world practices (where background noise exists and targeted behaviors potentially overlap), SideNet showed significant improvements compared to even the strongest baseline (i.e., ResNet): in addition to an average accuracy increment of 6.1%, the kernel model of our engine also caused much fewer false alarms.

Given that SideNet operates based on information gathered solely within its own user space, it does not require rooting the protected device or modifying semantics of any other apps running on the device. Consequently, SideNet could work on any existing Android distribution, while applying such a defense is as simple as installing a new app. Together with its high accuracy and low overhead, *SideNet could therefore help enhancing software-layer security of CPS and IoT environments involving Android devices, especially the industrial-based CPSs, by providing a cost-efficient supplement to potential malware mitigation measures.*

In summary, this paper makes the following contributions:

- This work is the first to propose the idea of leveraging system-wide side-channel data to detect sensitive app behaviors on Android for security purposes, which also makes the proposed detection approach suitable to industrial-based CPSs for being non-intrusive and therefore feasible to be applied without substantial extra costs.
- To construct the analysis kernel of SideNet, the authors propose a novel deep neural network model for Time Series Classification (TSC), which inherits the advantages of both ResNet and Encoder, the state-of-the-art practices for such tasks. The proposed model also demonstrated satisfying performance, which is important for industrial-

based CPSs where requirement on the stability of devices and running services is high.
- The authors have conducted in-lab simulations to evaluate SideNet, which suggested good effectiveness and performance, making it suitable to be applied in resource-limited environments such as industrial-based CPSs.

In the rest of this paper, we first present our problem statement and the threat model used in Section 2. The architecture, model design, and implementation details of SideNet are then discussed in Section 3. Procedures and results of the in-lab simulations on the proposed system are presented in Section 4. Finally, the conclusion of this paper is given in Section 5.

2 Problem Statement and Threat Model

Throughout this paper, we refer to Android devices in an assumed subject CPS as the "endpoints", and denote app behaviors that require dangerous permissions of Android, e.g., using camera, accessing geo-location, as "sensitive behaviors". Accordingly, the threat model considered in this paper involves adversaries attempting to inject malware instances into the endpoints of an assumed CPS. The goal of the proposed surveillance engine, on the other hand, is to help defending against such adversaries by providing supplementary information (specific to this paper, selected side-channel statistics) that can be used to infer what sensitive behavior had been executed on the monitored endpoints, which could potentially be used to analyze if the device had been infected by malware.

Considering practical requirements of industrial information systems, e.g., cost control, a surveillance engine suitable for the aforementioned application scenario should satisfy the following requirements:

- It does not require the underlying system framework to be modified in order to support it, otherwise all Android devices serving the protected CPS must be customized, which is not cost-efficient.
- It only requires normal user privilege.
- The proposed surveillance engine itself should not be intrusive, i.e., its information gathering operation should not violate any access control policy enforced by Android, such as app sandboxing.

Meanwhile, considering the context of industrial information systems, we limit the capability of adversaries to a practical extent as well. That is, they are assumed to have no physical access to any part of the CPS under attack, and the malware instances could only be injected remotely.

3 The SideNet Engine

The surveillance engine proposed in this paper, i.e., SideNet, works by periodically collecting side-channel statistics from the monitored endpoints on-the-fly. The obtained data is then analyzed using a deep learning model to determine whether and which sensitive behavior had happened. Typically, an endpoint in an industrial-based CPS would be running a relatively stable set of apps, and is therefore likely to demonstrate a consistent pattern of sensitive behaviors during regular operations. As a result, by aggregating logs of sensitive behaviors reported by SideNet from all the monitored devices, the subject CPS would then be able to verify the pattern of each individual device and consequently detect "abnormal nodes" that may have been compromised by malware.

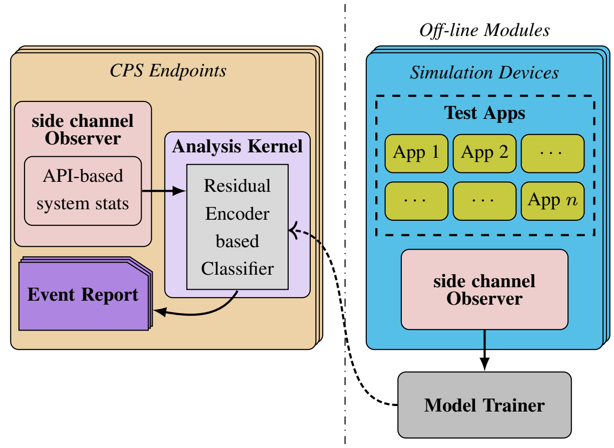


Fig. 1: The general architecture of SideNet.

Fig. 1 illustrates the general architecture in which SideNet is deployed. On the monitored endpoints, SideNet consists of a side-channel observer which collects statistical side-channel information, as well as an analysis kernel which takes these side-channel data as input to detect sensitive behaviors. Meanwhile, a group of simulation devices execute as off-line components of SideNet and collect ground truth samples for training the analysis kernel. These simulation devices cover different models of the subject endpoints, and are installed with the same version of Android framework as well (this is possible for CPS where usually only limited types of hardware and operating system configurations are involved). In addition, the simulation devices also have the same side-channel observer deployed and a variety of test apps (though these apps might be different from those installed on the endpoints) running simultaneously to generate side-channel footprints for model training. Constructed in this way, the simulation devices provide runtime environments as close to those on the monitored endpoints as possible, providing credibility for the collected ground truth samples.

3.1 Inputs to SideNet

According to the assumed application scenario, the authors emphasize the requirement that the collection of the statistical data, i.e., inputs to SideNet

- Does not require root privileges or the underlying Android system to be modified in anyway; and
- Does not require access or tamper with either modules of the system or code of other user apps.

It is also considered that any resources accessible to a normal user app, including those requiring permissions, are available and potentially part of the inputs to SideNet. Given that statistical side channels are retrieved by either calling specific APIs or accessing the `procfs` files, and that retrieving any statistics from a `procfs` file requires the observer of SideNet to parse the entire file (and therefore inefficient), if information provided by a `procfs`-based side channel is of the same category as an API-based one, SideNet is designed to use the API-based source for better efficiency.

Table 1 presents a verified list of statistical side channels that meet the aforementioned requirements and are available in Android 9 and 10. After multiple rounds of patching and updating

TABLE 1: List of verified API- and `/procfs`-based side channels in Android 9 and 10 that meets the requirement of SideNet.

Name of side channel	Type	Category
`storageManager.getAllocatableBytes()`	API	memory allocation
`storageManager.getCacheQuotaBytes()`	API	memory allocation
`storageManager.getCacheSizeBytes()`	API	memory allocation
`File.getFreeSpace()`	API	memory allocation
`File.getUsableSpace()`	API	memory allocation
`Process.getElapsedCpuTime()`	API	CPU usage
`TrafficStats.getMobileTxBytes()`	API	network traffic
`TrafficStats.getTotalTxBytes()`	API	network traffic
`TrafficStats.getMobileTxPackets()`	API	network traffic
`TrafficStats.getTotalTxPackets()`	API	network traffic
`TrafficStats.getMobileRxBytes()`	API	network traffic
`TrafficStats.getTotalRxBytes()`	API	network traffic
`TrafficStats.getMobileRxPackets()`	API	network traffic
`TrafficStats.getTotalRxPackets()`	API	network traffic
`/proc/meminfo`	`procfs`	MemAlloc
`/proc/net/sockstat`	`procfs`	NetTraff
`/proc/net/protocols`	`procfs`	NetTraff

of the Android system, remaining side-channel resources of this type provide measurements mainly on memory allocation, network traffic, as well as CPU usage. In addition, for each of the remaining `procfs`-based side channels, there exist API-based ones which provide measurements of the same category. These suggest that `procfs`-based side channels contribute little (if any) in addition to API-based ones in obtaining the demanded side-channel profile in the latest Android systems. Intuitively, footprints of memory allocation and network traffic could be valid signatures of sensitive app behaviors. For example, when an app uses the camera, a buffer is immediately allocated for maintaining the preview picture. Similarly, when an app requests the device's location information, a specific set of network protocols are involved subsequently, causing a pattern of network traffic due to the particular packets being sent and received. Motivated by this intuition, the authors adopt the API-based side channels given in Table 1 as the measurement vector of SideNet for detecting sensitive behaviors.

3.2 Side-channel information observer

After determining the specific side-channel information to use, we face another challenge in building SideNet to achieve *long-term* surveillance. Recall that SideNet is built to periodically collect readings of API side channels, i.e., it maintains a loop in which all APIs listed in Table 1 are invoked one by one without pausing. Therefore, to achieve long-term surveillance, this loop needs to be kept executing incessantly. The simple idea of creating a "long-lived" service to run such a loop would not work, as the service will be stopped by Android after a certain amount of time (e.g., after the device goes to sleep). Meanwhile, since SideNet demands a high sampling rate, approaches like using `AlarmManager` for periodic sampling is also not an option considering the minimum period of such an alarm (15 minutes) is still too long for the assumed tasks.

To overcome this problem, SideNet adopts a robust and persistent side-channel scanning process leveraging multi-thread programming as well as the `JobScheduler` mechanism of Android. Specifically, SideNet's side-channel observer continuously creates worker threads to retrieve the side-channel statistics. Immediately after such a worker thread is created, it schedules a job to iteratively retrieve the demanded side-channel readings for a predetermined number of rounds (set to 1000 in our proof-of-concept demo), and transmits the collected data to SideNet's analysis kernel in real time using the anonymous shared memory (ashmem) mechanism of Android. After that, the current worker thread creates a new worker thread, and then ends the job it scheduled and terminates itself. The new worker thread will then schedule a new job to continue the data collection in the same way as was done in the previous thread. Built in this way, none of the worker threads (and consequently the jobs they schedule) would run for too much time such that it gets killed by the system before it terminates normally, allowing SideNet to provide the desired long-term surveillance.

3.3 Residual Encoder based Classifier

In identifying sensitive app behaviors, the side-channel observer transmits the statistics it retrieves in the form of a stream to its analyzer. The analyzer slices the data stream into fixed-length time series using a sliding window and performs classification on these time series. This is formalized as a learning task for solving a multi-class time series classification (TSC) problem. Given $(X_i, y_i), i \in \{1, 2, \cdots, m\}$, where X_i is a K-dimensional time series, i.e., $X_i = [X_i^1, X_i^2, \cdots, X_i^K]$ with each $X_i^j = [x_1^{ij}, x_2^{ij}, \cdots, x_T^{ij}]$ being an ordered sequence of readings (of length T) from a specific side channel reading. $y_i \in Y = \{Y_1, Y_2, \cdots, Y_N\}$ indicates which sensitive app behavior X_i corresponds to. Note that SideNet identifies the sensitive app behaviors on top of making the binary decision "whether a sensitive behavior had occurred". Therefore, we use $Y_1, Y_2, \cdots, Y_{N-1}$ to represent different sensitive behaviors while using Y_N to represent the situation of "none of these behaviors occurred". With $X_i \in X$ denoting the collection of K-dimensional time series corresponding to all types of sensitive behaviors in Y, the goal of SideNet is to learn a mapping from X to Y, $f : X \mapsto Y$, to predict whether an unseen time series collected by the side channel observer indicates a sensitive app behavior; and if so, to determine what type of behavior is represented by the time series. f can be learned by minimizing the following objective function:

$$\min_f \sum_i \mathcal{L}(f(X_i), y_i) + \lambda\Omega(f), \tag{1}$$

where $\mathcal{L}(\cdot,\cdot)$ is the empirical loss, $\Omega(f)$ is a regularization term imposed on the prediction function, and λ is the trade-off parameter to be tuned.

The design of the classifier is crucial to the effectiveness of SideNet due to the following reasons.

- SideNet supports online detection, and therefore demands real-time performance which accordingly requires the overhead of its classifier to be as low as possible;
- Furthermore, the mechanism of `JobScheduler`, together with hardware acceleration features of the subject endpoint devices, inevitably introduce jitter into SideNet's side-channel sampling process, bringing additional difficulties to the classification task on time series constituted by samples affected by such jittering.

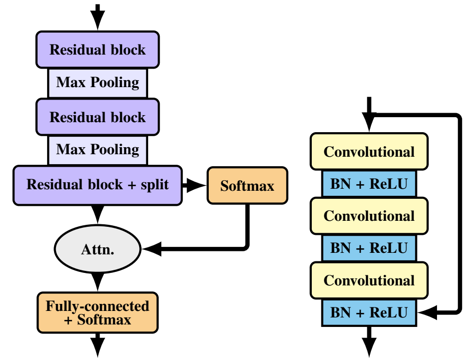
The traditional TSC approaches, which are in essence probability statistical models [3], [14], usually involve a computationally intensive pre-processing phase, making them undesired choices. Thus in this paper, the authors choose to build a deep learning model as the kernel of SideNet. Note that *the main value of proposing such a deep learning model is about enabling accurate and high-performance identification of potential sensitive app* *behaviors reflected by side-channel time series, which therefore indirectly contributes to the security of industrial-based CPSs via the overall functionality of SideNet.*

Latest studies [8] had shown that the residual learning framework, or ResNet [9], was considered the best practice compared to other existing deep learning models for TSC tasks. This is mainly due to its deep and flexible architecture as well as the residual connections which help to reduce the vanishing gradient effect. In fact, it is believed that the high generalization capability of deep CNNs on the TSC tasks is rather natural considering the good performance of similar models which learn on two dimensional spaces. As explained in Section 1, there are two major challenges in addressing the TSC task of the specific application scenario of this paper, both of which indicate that generalization capability of the deep learning model used in SideNet is rather critical.

Meanwhile, a key insight we have on the side-channel pattern of sensitive app behaviors is that, *when represented as time series, each sample (i.e., the aforementioned vector of side channel readings) within the series is of different importance*. This brought out another recent TSC method referred to as the Encoder [16]. By replacing the GAP layer of a standard convolutional network with an attention layer, Encoder is made capable of learning the importance of each time stamp in a multi-dimensional time series. However, the standard design of Encoder only contains 3 simple convolutional blocks, which makes its depth inferior compared with other state-of-the-art deep learning models. Considering that previous experiences [8], [9] strongly suggested that "going deeper" is the correct path towards building better deep learning models (regardless the type of target task), this paper aims to inherit the advantages of both ResNet and Encoder. Towards such a direction, a hybrid (and improved) deep neural network model called Residual Encoder based Classifier is proposed as the analysis kernel of SideNet.

The general architecture of Residual Encoder based Classifier is illustrated in Fig. 2. On top of the conventional design of Encoder, all convolutional blocks of the network are replaced with residual blocks as in ResNet to obtain high generalization capability while preserving Encoder's feature of being able to learn the importance of individual samples within the time series. The current prototype of SideNet used three residual blocks in building this analysis kernel, each built by stacking 3 convolutional blocks consisting of a convolutional layer followed by a batch normalization layer and a ReLU activation layer, with the shortcut connection added to link the residual block's output to its input. The number of filters in the residual blocks are respectively set to $\{64, 128, 128\}$, with the convolution operation fulfilled by three 1-D filters of sizes $\{9, 5, 3\}$ without striding. Note that these configurations are set to be identical to the architecture reported in a recent study [24] merely to make the performance comparison between the two models more convincing. It's necessary to emphasize that the aforementioned configurations is only a proof-of-concept demonstration, and it is expected that the kernel of SideNet be built deeper.

Finally, note that being a deep learning based system, SideNet is expected to be updated periodically such that behaviors of the emerging new apps (and malware) be learned in time. This, however, leads to a subtle dilemma: on one hand, SideNet needs an analysis kernel built as deep as possible in order to achieve better accuracy; on the other hand, deeper (and consequently larger) structures in return result in larger overhead and network traffic cost during model updating. Nowadays, deep neural networks



(a) General structure (b) A residual block

Fig. 2: Flowchart of the proposed Residual Encoder based Classifier and its residual blocks.

could easily take a memory space of several megabytes, which means that updating them periodically can be equivalent to downloading and installing a new app every few minutes, and therefore is practically unacceptable. To address this problem, the authors resort to sparse learning [6], a recently proposed accelerated training approach to help constructing SideNet's analysis kernel.

Sparse learning leverages exponentially smoothed gradients to identify layers and weights which reduce the error efficiently, and is able to achieve over 5x faster training while significantly reducing the number of valid weights in the resulting networks [20]. According to the definition of sparse learning, by setting the training process to reduce the weights of SideNet to $n\%$ of a normal dense neural network, the network flow required for updating the resulting analysis kernel could be reduced by $(100 - n)\%$. Currently we set $n = 20$ in the proof-of-concept construction of SideNet, i.e., the updating cost is 80% less than a dense model.

4 Evaluation

A number of in-lab simulations have been carried out to evaluate the capability of SideNet, in particular, its accuracy in identifying sensitive app behaviors as well as its performance cost. Eight dangerous permissions belonging to seven categories were selected for the simulations, namely `ACCESS_COARSE/FINE_LOCATION`, `CAMERA`, `RECORD_AUDIO`, `READ_SMS`, `READ_CONTACTS`, `READ_CALL_LOG` and `READ_CALENDAR`. The authors focused on these particular types of behaviors because they have been confirmed to be no longer detectable using the existing non-intrusive detection approaches [23]. Specifically, sensitive behaviors related to `ACCESS_COARSE/FINE_LOCATION` and `CAMERA` permissions were once detectable by analyzing interrupt records provided in `/proc/interrupts`. However, access to this `procfs` file has been blocked since Android 9 in an attempt to stop side-channel attacks. Behaviors related to the other selected permissions, on the other hand, were originally not covered. Therefore, demonstrating that the SideNet engine be capable of detecting these sensitive behaviors with high accuracy would be of the most significant impact in challenging the common belief in the security community (that the system-wide statistical side channels do not disclose private information of individual apps). Furthermore, it's also worth mentioning that some of the tested permissions should be of high relevance to actual Android-related attacks on real CPSs. Specifically, sensor-based privacy theft attacks typically deploy camera- and microphone-based Trojan horses on Android devices to stealthily capture and send video/audio to attackers, and hijacking the SMS receiving process is a popular way of implementing the control panel of such attacks [12]. A systematic study on malicious applications targeting Android-based mobile CPS had also pointed out that requesting location information and reading SMSs are two of the top 12 features identified in malware samples [21]. It is natural to extend SideNet to cover more sensitive app behaviors, which is left as future work.

Recall that SideNet uses off-line simulation devices to collect ground truth samples to facilitate model training, which consist of the side-channel time series, the types of sensitive behaviors taking place, as well as the corresponding time stamps. To this end, we make use of `monkeyrunner` to automatically interact with test apps and log time stamps of the resulting sensitive behaviors. As for those sensitive behaviors not triggered directly via user interactions, e.g., behaviors requesting coarse- and fine-grained geo-locations, a customized Android Open Source Project (AOSP) system with additional logging capability built into the framework was used to collect ground truths. Note that this solution restricts Android framework modification to our in-lab simulation devices only, while the monitored endpoints are not required to have any modification to the Android system.

4.1 Experimental setup to evaluate effectiveness in detecting sensitive behavior

Two rounds of classification simulations were conducted in order to understand the effectiveness of SideNet more comprehensively. The first simulation assumed an ideal scenario, in which SideNet as well as the baselines were used to identify sensitive app behaviors that were intentionally *isolated* to avoid having their side channel patterns being interfered by other program activities. The goal of such an idealized simulation was to understand the optimal setting of the length of side channel time series (i.e., the value of T as defined in the beginning of Section 3) for SideNet.

The second simulation focused on a more practical scenario where the targeted sensitive behaviors were triggered in parallel with random timings, such that some of them could become *overlapping* with each other. Particularly, the authors want to emphasize the practical importance of considering the second scenario, because it is possible for multiple apps to initiate different types of sensitive behaviors within a very small time window. Given that the side channels adopted are all system-wide statistics, observable pattern of a sensitive behavior launched by one app could therefore be affected by those of other app components. As such, *in evaluating the effectiveness of SideNet, the second simulation was considered as the primary indicator, while the first one was seen only as an auxiliary evidence*.

In both simulations presented here, a number of commercial apps were selected as test subjects, each of which presents behaviors related to at least one of the seven highlighted dangerous permissions. The test subjects were then run on a total of three test devices (all of which are Google Pixel 3 XL) under the manipulation of `monkeyrunner`, which allows our Python script to execute and interact with a subject app (e.g., sending screen touches and keystrokes to it). An automatic Python script was used to induce the sensitive behaviors in the app and collect ground truth information including time stamps at which the sensitive behaviors are triggered. Side-channel observer runs side-by-side on these simulation devices to retrieve side-channel time series.

TABLE 2: Test subjects and background apps of our experiments.

Test and background apps	Permission
`Google Camera`, `Twitter`, `Snapchat`	`CAMERA`
`Audio Recorder`, `Easy Voice Recorder`, `Voice Recorder`	`RECORD_AUDIO`
`Google Map`, `Grab`, `iRIS`	`ACCESS_COARSE/FINE_LOCATION`
`Google Messages`, `Promessage`, `Messages OS 12`	`READ_SMS`
`Whatsapp`, `Wechat`, `Line`	`READ_CONTACTS`
`CallApp`, `Contacts+`, `Google Hangouts`	`READ_READ_CALL_LOG`
`Google Calender`, `DigiCal`, `One Calender`	`READ_CALENDAR`
`Google Mail`, `Lazada`, `Spotify`, `CNN`, `Fox News`, `BBC News`, `Guardian`, `Today`, `Times of India`, `Flipboard`	none (background)

In addition, considering that it is unlikely in realistic scenarios to have a test subject app running as the one and only active app on a device, a group of other apps which receive opportunistic push notifications (e.g., news apps, e-commercial apps) were kept running in the background, such that the passive behaviors of these apps would produce realistic noise that a real user device would have. Table 2 presents the list of test subjects and background apps used in this experiment. Note that some of the background apps request the device's current location on startup, which are ignored in our experiments. Also, in order to make the behavior of `Google Mail` more realistic, the authors registered a test Gmail account for a number of subscriptions from websites such as McDonald®, and let another script-controlled email account send messages to this Gmail address at a random interval between 20 and 40 minutes.

It is necessary to emphasize that although it was common commercial apps for smartphones that were included as subjects, *configuring the experiment in this way does not undermine the validity of the experiments given that the effectiveness of SideNet is never supposed to be limited by the type of apps running on the endpoints.* In addition, intuitively, there are two portions of executions contributing to the resource usage signature of a sensitive app behavior, namely methods of the user app related to the behavior, and methods of the Android framework involved in the service requested by the behavior — and it's not hard to see that the system-side execution, which is generally the same for all apps requesting the same API, contributes more significantly to the consistency of such behavioral signatures. As such, a sufficiently trained SideNet model would be able to safely generalize to different Android subversions as long as such a target subversion provides the same set of framework APIs to user apps. In fact, we argue that our experiment configuration would actually make the detection of sensitive app behaviors harder than in most real-world scenarios with CPSs which tend to involve more dedicated devices and therefore could have fewer background apps running. The generality of SideNet is further discussed in Section 4.5

The performance of SideNet was compared with a number of baseline learning methods, including LSTM, MLP, FCN, Encoder, and ResNet. The authors used a basic LSTM model containing a single LSTM layer of 128 hidden units to provide a common bottom line for the simulations. For MLP, FCN, and ResNet, the authors used the same configuration as used in the related work [24] in order to provide a fair comparison between the baselines and SideNet. Similarly, the Encoder model used in the simulations also adopted the same configuration as in a previous report [16].

4.2 Simulation 1: Isolated sensitive behaviors

In this simulation, each test subject app was run for a period long enough to capture 1,000 samples of time series readings of the sensitive app behavior it corresponds to. The gap between any two neighboring time series was set to a random value between 1 to 5 minutes. The long timing gap between every two events ensures that the sensitive behaviors are *isolated* in the sense that their patterns of side-channel information do not interfere with each other. Furthermore, in order to enable detection of non-sensitive events — side channel patterns that correspond to the absence of sensitive behavior — a stream of baseline data for a period of 5 hours were collected by having the test devices work only with background apps (i.e., no sensitive behavior is invoked during this period).

4.2.1 Length of the time series

The authors compared the performance of all tested models in handling time series of 5 different T values between 20 and 100. This is chosen based on the statistics on the timing of all collected samples, which suggested that the average duration between two consecutive probing is around 50ms. Therefore, the 5 selected T values could give us time series covering time periods of about 1 to 5 seconds, which are enough to represent the execution routine of a specific app behavior completely. The authors did not move to larger T because SideNet can only start working on a potential detection of sensitive behavior after all samples of a segment of side channel statistics (i.e., a complete time series) has been retrieved. Therefore, adopting time series of a larger T will result in a long delay in the detection. Considering that using time series of length 100 already translates to a delay of around 50ms$\times$50=2.5s, testing on $T > 100$ is therefore not in line with the goal of dynamic real-time detection.

4.2.2 Accuracy of SideNet vs. baselines

Out of all the time series collected in this simulation, the authors used 10-fold validations to test the performance of both SideNet and the baselines for each selected T. The overall accuracy of the tested models for these different settings are presented in Fig. 3. We can see that accuracy of most tested models went down with decreasing value of T (except for MLP, in which the accuracy sightly increases before T is decreased to 60), with the most significant drop from 40 to 20 (except, again, for LSTM, in which the most significant accuracy drop came earlier). This suggested that for many sensitive app behaviors, their side channel patterns cannot be effectively profiled by time series of 20 samples, hence using time series of a minimum of 40 samples is a reasonable setting to minimize the latency caused by retrieving side channel statistics while achieving high identification accuracy. Meanwhile, it is also worth mentioning that for each selected value of T, SideNet outperformed all the tested baselines with accuracy greater than 95%. Although the strongest baseline, namely ResNet, had also achieved high accuracy, the authors argue that this result alone does not undermine the significance

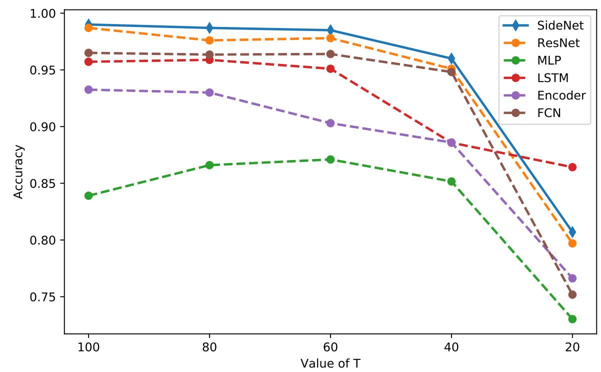


Fig. 3: Classification accuracy over different time series lengths.

of the proposed model since the simulated scenario is highly idealized, and hence it's not surprising to see a state-of-the-art model (especially the one serving as one of the bases of SideNet) also performs well.

4.2.3 Importance of individual side channels and groups of side channels

In addition, given that the side channels adopted by the proposed engine are essentially categorized in 3 groups, i.e., those related to memory allocation, CPU usage, and network traffic, the authors further studied how each individual adopted side channel and how the 3 side-channel categories affect the performance of the proposed engine. This is done by investigating a series of modified SideNet models, each of which either had one of the side-channel APIs listed in Table 1 omitted, or had an entire group of them omitted (note that the category of CPU usage consists of only one API). As shown in Fig. 4, the removal of network-traffic-related side channels caused the most significant drop in F1-scores of the resulting model, while the impact of removing memory-allocation-related side channels took the second place. On top of that, the impact of removing any complete group of side channels was more significant than that of removing an individual one. These suggested that network traffic signature could be the most important feature when it comes to identifying sensitive app behaviors, and memory allocation footprints contributed only slightly less. In addition, multiple side channels which reflect the same type of profile could be enhancing the effectiveness of each other and increasing the overall robustness of SideNet.

4.3 Simulation 2: Overlapping sensitive behaviors

In the second simulation, the authors tried to understand the performance of SideNet in a more complicated scenario in which sensitive behaviors were originated from multiple apps (including potentially malicious ones) at random timings. That is, we simulated the scenarios where multiple apps respectively carry out sensitive behaviors at their own pace, making it possible for the timing gaps between those behaviors to be very small (in which case the side channel patterns of these behaviors could partially overlap).

Note that on Android 9 and above, apps running in the background can no longer access camera or microphone of the device. Therefore, behaviors related to `CAMERA` and `RECORD_AUDIO` permissions are guaranteed to be originating from foreground operations. This also implies that operations related to these permissions would never overlap with each other. Sensitive operations

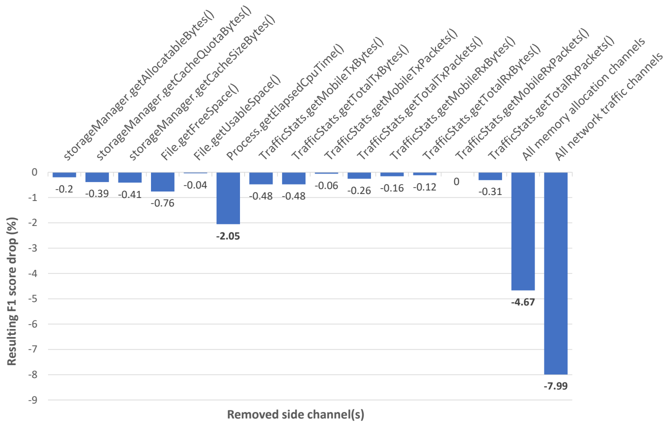


Fig. 4: Performance impact on SideNet when removing an individual side channel (group).

from the foreground apps can only be triggered one by one, while those from the background apps can be raised more frequently. Taking these observations into account, the test environment of this simulation is configured in the following way:

- For the CAMERA and READ_CALENDAR permissions, the same test subjects (i.e., commercial apps listed in Table 2) as used in the Simulation 1 were adopted to generate the related sensitive operations from foreground;
- The RECORD_AUDIO permission was omitted because its behavior cannot be overlapped with CAMERA, which does not serve the purpose of our Simulation 2;
- Behaviors requiring the remaining five highlighted permissions were conducted by a customized test app triggering sensitive behaviors from the background.

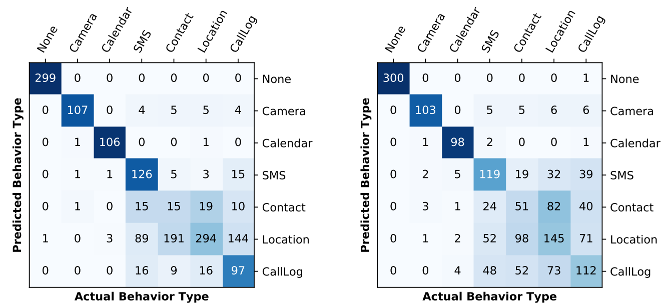
The authors find this configuration especially interesting because the commercial apps were controlled via ADB (Android Debug Bridge) and therefore mimic real user interactions with the device, while the customized test app works purely in the background and thus mimics the behavior of malware trying to eavesdrop user private information in a stealthy manner. The interval between two neighboring events related to the same permission were set as a distinct prime number, namely 23 seconds each for operations related to CAMERA and READ_CALENDAR permissions, 17 seconds for those related to READ_SMS, 13 seconds for READ_CALL_LOG, 11 seconds for READ_CONTACTS, and 7 seconds for ACCESS_COARSE/FINE_LOCATION. This would likely provide cases with different timing gaps between two different types of sensitive behaviors after a sufficiently long period of time.

4.3.1 *Overall accuracy of SideNet vs. baselines*

Once again, this simulation used 10-fold validations to test the performance of both SideNet and the baselines, of which the results (include overall accuracy, recall, and f1-score) are shown in Table 3. The results show that when the simulated scenarios got closer to real-world practices, accuracy of all tested models dropped compared to that in Simulation 1 due to the severe noise from overlapping app behaviors. At the same time, the authors also observed that SideNet now significantly outperforms all competing models including ResNet, demonstrating an accuracy improvement of 6.1% against this strong baseline. This, together with the results from Simulation 1, suggested that compared to the existing models, SideNet demonstrates better robustness when it is applied in real-world applications.

TABLE 3: Average performance of selected approaches in identifying potentially overlapping sensitive app behaviors.

Approach	Accuracy	Macro Recall	Macro f1
LSTM	0.560	0.588	0.563
MLP	0.530	0.560	0.557
FCN	0.561	0.604	0.592
Encoder	0.579	0.630	0.632
ResNet	0.585	0.632	0.631
SideNet	**0.646**	**0.661**	**0.671**



(a) Result of SideNet (b) Result of ResNet

Fig. 5: Heatmaps for results of SideNet/ResNet (Simulation-2).

4.3.2 *Accuracy of individual sensitive behavior*

In addition, the authors had a closer look into the performance of both ResNet and the proposed model regarding each specific sensitive behaviors as well as the non-sensitive samples (which correspond to no sensitive behavior at all). The results are shown as heatmaps in Fig. 5 (where the label "none" indicates the non-sensitive category, i.e. the test devices were to work only with background apps with no sensitive behaviors occurred). For both models, precision in identifying camera-, calendar- and SMS-related behaviors were still high, while those for the remaining cases dropped significantly. The recall of both models in identifying SMS-, call-log-, and contact-related behaviors also received significant negative impacts, although that for SideNet in identifying locations wasn't affected as badly as in ResNet.

The authors believe that this is most likely due to the difference between the customized single test app in simulating sensitive operations and those in commercial apps. Specifically, the sensitive operations simulated by our test app involved no UI display or network communication with remote entities, i.e., the digital footprints of these operations are designed to be very obscure, making them easy to be confused with background noises or other passively triggered sensitive behaviors (like location requests). Another reason for the misclassifications among these behaviors is that SMS and contacts, although categorized as different sensitive behaviors, are retrieved by user apps in very similar ways via a *content provider*. The reading of these types of data, although requiring different permissions, is carried out similarly in the form of SQL queries using a unanimous API (ContentResolver.query()). This makes app behaviors involving content reading intrinsically similar due to the shared low-level execution routine. Furthermore, note that many observed

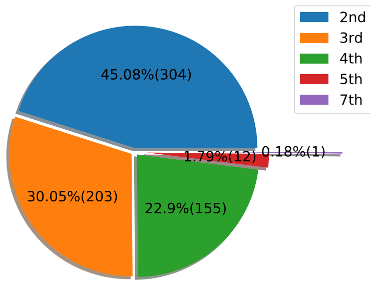


Fig. 6: Statistics of SideNet on the ranking of actual behavior type for misclassified samples.

misclassifications involved recognizing other sensitive operations as location requests, which is the most frequently triggered operation (once per 7 seconds as mentioned above) and therefore overlapped with other sensitive operations in many samples of our training set.

4.3.3 Ranking of ground truth in misclassified cases

The authors further looked into the higher-order statistics of SideNet's performance, namely, the ranking of ground truths when a sample was misclassified. As shown in Fig. 6, over 75% of SideNet's misclassifications had the ground truth of the sample ranked at either the second or the third place. This indicates that more often than not, mistakes made by the proposed model are still close shots.

4.3.4 Other observations

There were also a few other observations out of the above results. First, the comparison between SideNet and the standard Encoder design suggested that "going deeper" had contributed significantly in improving the accuracy of neural network models. Next, the comparison between SideNet and ResNet indicated that for the application scenario of sensitive app behavior identification, the authors' intuition as mentioned in Section 3.3 did make sense, i.e., individual time stamps within a time series of side channel statistics could actually be of different importance. As a result, building the SideNet engine on top of the attention mechanism of Encoder indeed improved its performance. Moreover, from the details shown in Fig. 5, the authors believe that applying the attention mechanism had also made SideNet more "prudent", i.e., it makes conservative mistakes rather than aggressive ones (which is actually a desirable feature of the technology). These together made SideNet a better solution in addressing the three challenges of the application scenario assumed in this paper (as stated in Section 1) compared with the existing baseline methods. The authors believe that these readings can also be generalized to other application scenarios which share similar characteristics.

4.4 Performance Cost

Being an app behavior surveillance tool indicates that SideNet is supposed to be kept in a long-term operating status. As a result, to understand whether it is efficient enough for real CPSs (especially those in industrial information systems), the performance of SideNet needs to be evaluated from two aspects: the computational cost of the deep learning model as its analysis kernel, as well as the power consumption per hour for running the whole system.

TABLE 4: Power consumption (in percentage to the total battery capability) of selected apps for an 1 hour running test.

App	Power consumption
Twitter	0.03%
BBC News	0.04%
Fox News	0.01%
Google map (idle)	0.03%
Google map (navigation)	1.14%
SideNet	**0.07%**

With regard to computational cost, SideNet takes 0.4MB for Parameters and its amount of MAdds is 27.7M. In comparison, readers may refer to deep learning application known as MobileNet[2], which is designed specifically for mobile devices: the 3rd (and latest) version of this model takes up to 5.4MB for Parameters and has an amount of MADDs of up to 217M. This suggests that the deep learning model of SideNet is suitable for mobile devices.

With regard to power consumption, SideNet was tested on a Google Pixel 3 XL with a total of 4 other popular apps (namely, Google maps, Twitter, BBC News, and Fox News) running together, and we measured the power consumption of these apps during a one-hour period using Battery Historiana toolkit provided by Google for inspecting battery related information and events. Specifically, two rounds of tests were performed. In the first round of experiment, all tested apps other than SideNet were simply started and left idle in the background. In the second round, Google maps was turned into navigation mode and kept navigating during the entire period of the experiment, simulating a common and reasonable high power consumption use case of Android apps. Table 4 shows that although SideNet was kept running during the whole experiment, its power consumption was only slightly higher than the idle apps, and was significantly lower than the navigating Google maps (the latter consumed 16 times more power than SideNet when navigating). Therefore, the authors feel safe to argue that SideNet is suitable to be deployed in real industrial-based CPSs.

4.5 Generality of SideNet

As discussed in Section 4.1, the applicability of SideNet can be expected as long as the targeted Android subversion provides a consistent set of framework APIs that include what SideNet uses. By analyzing the design and source code of Android extensions for embedded systems, namely Android Things and Android Automotive hardware abstraction layer (HAL), we have verified that both extensions satisfy this requirement, and SideNet runs smoothly in both systems. Our investigation shows that these extensions generally work in the form of extra libraries or software layers in supportive of the Android core framework. Therefore,

- Any sensitive app behavior conducted on such embedded extensions of Android still requires executing the same related services in the Android core framework, and
- All APIs which SideNet relies on (as listed in Table 1) are still available.

This feature of Android's architecture clearly indicates that when applied in real-world CPS systems, SideNet remains effective as demonstrated in the above simulations.

2. https://github.com/tensorflow/models/tree/master/research/slim/nets/mobilenet.

5 Conclusion

In this paper, a novel deep-learning-based engine, called SideNet, has been proposed for detecting sensitive behavior of Android apps dynamically and non-intrusively. Based on a state-of-the-art TSC model called Encoder, SideNet adopts residual learning to replace the simple convolutional blocks and hence achieves better generalization capability while maintaining key advantages of Encoder, i.e., being able to learn the importance of individual time stamps of a time series. In addition, sparse learning was introduced into the training of SideNet, which helps increase the efficiency of updating a running analysis kernel of SideNet. Evaluations showed that SideNet significantly outperformed a number of strong TSC baselines when identifying sensitive app behaviors in the test designed to simulate practical scenarios where app behaviors could potentially overlap with some others partially with regarding to their timings. Specifically, the proposed engine achieved an accuracy improvement of 6.1% compared to the strongest baseline, ResNet. The authors believe that this work opens a new door to developing more comprehensive behavior monitoring techniques on Android.

Acknowledgment

This research/project is supported by the National Research Foundation, Singapore under its AI Singapore Programme (AISG Award No: AISG-100E-2018-004), National Natural Science Foundation of China (Grant No. 61702399 and 61972215) and National Key R&D Program of China (2018YFA0704703).

References

[1] "VirtualApp: Virtual Engine for Android," https://github.com/asLody/VirtualApp, 2018.

[2] A. Alostad, G. Seraj, F. Malallah, J. Alhajri, H. Ali, M. Aloqaily, and A. Bani-Mustafa, "An application to manage widespread social media accounts with one smart touch," in *2018 Fifth International Conference on Software Defined Systems*. IEEE, 2018, pp. 176–181.

[3] A. Bagnall, J. Lines, A. Bostrom, J. Large, and E. Keogh, "The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances," *Data Mining and Knowledge Discovery*, vol. 31, no. 3, pp. 606–660, 2017.

[4] O. Bouachir, M. Aloqaily, L. Tseng, and A. Boukerche, "Blockchain and fog computing for cyberphysical systems: The case of smart industry," *Computer*, vol. 53, no. 9, pp. 36–45, 2020.

[5] S. K. Dash, G. Suarez-Tangil, S. Khan, K. Tam, M. Ahmadi, J. Kinder, and L. Cavallaro, "Droidscribe: Classifying android malware based on runtime behavior," in *2016 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2016, pp. 252–261.

[6] T. Dettmers and L. Zettlemoyer, "Sparse networks from scratch: Faster training without losing performance," *arXiv preprint arXiv:1907.04840*, 2019.

[7] W. Diao, X. Liu, Z. Li, and K. Zhang, "No pardon for the interruption: New inference attacks on android through interrupt timing analysis," in *Proc. of the 2016 IEEE Symposium on Security and Privacy*. IEEE, 2016, pp. 414–432.

[8] H. I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller, "Deep learning for time series classification: a review," *Data Mining and Knowledge Discovery*, vol. 33, no. 4, pp. 917–963, 2019.

[11] J. Lee, M. Azamfar, and J. Singh, "A blockchain enabled cyber-physical system architecture for industry 4.0 manufacturing systems," *Manufacturing Letters*, vol. 20, pp. 34–39, 2019.

[9] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. of the 29th IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[10] I. D. Kazakov, N. L. Shcherbakova, A. Brebels, and M. V. Shcherbakov, "Accelerometer data based cyber-physical system for training intensity estimation," in *Cyber-Physical Systems: Advances in Design & Modelling*. Springer, 2020, pp. 325–335.

[12] L. Lei, Y. Wang, J. Zhou, L. Wang, and Z. Zhang, "A threat to mobile cyber-physical systems: Sensor-based privacy theft attacks on android smartphones," in *2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*. IEEE, 2013, pp. 126–133.

[13] J. Li, L. Sun, Q. Yan, Z. Li, W. Srisa-an, and H. Ye, "Significant permission identification for machine-learning-based android malware detection," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 7, pp. 3216–3225, 2018.

[14] J. Lines, S. Taylor, and A. Bagnall, "Time series classification with hive-cote: The hierarchical vote collective of transformation-based ensembles," *ACM Transactions on Knowledge Discovery from Data*, vol. 12, no. 5, p. 52, 2018.

[15] A. M. Rahmani, T. N. Gia, B. Negash, A. Anzanpour, I. Azimi, M. Jiang, and P. Liljeberg, "Exploiting smart e-health gateways at the edge of healthcare internet-of-things: A fog computing approach," *Future Generation Computer Systems*, vol. 78, pp. 641–658, 2018.

[16] J. Serrà, S. Pascual, and A. Karatzoglou, "Towards a universal neural network encoder for time series." in *Proc. of the 21st International Conference of the Catalan Association for Artificial Intelligence*, 2018, pp. 120–129.

[17] L. Shi, J. Fu, Z. Guo, and J. Ming, ""jekyll and hyde" is risky: Shared-everything threat mitigation in dual-instance apps," in *Proc. of the 17th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, 2019, pp. 222–235.

[18] R. Spreitzer, F. Kirchengast, D. Gruss, and S. Mangard, "Procharvester: Fully automated analysis of procfs side-channel leaks on android," in *Proc. of the 2018 on Asia Conference on Computer and Communications Security*. ACM, 2018, pp. 749–763.

[19] R. Spreitzer, G. Palfinger, and S. Mangard, "Scandroid: Automated side-channel analysis of android apis," in *Proc. of the 11th ACM Conference on Security & Privacy in Wireless and Mobile Networks*. ACM, 2018, pp. 224–235.

[20] S. Srinivas, A. Subramanya, and R. Venkatesh Babu, "Training sparse neural networks," in *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2017, pp. 138–145.

[21] D. Su, J. Liu, W. Wang, X. Wang, X. Du, and M. Guizani, "Discovering communities of malapps on android-based mobile cyber-physical systems," *Ad Hoc Networks*, vol. 80, pp. 104–115, 2018.

[22] K. Tam, S. J. Khan, A. Fattori, and L. Cavallaro, "Copperdroid: Automatic reconstruction of android malware behaviors." in *Proc. of the 22nd Annual Network and Distributed System Security Symposium*, 2015.

[23] X. Tang, Y. Lin, D. Wu, and D. Gao, "Towards dynamically monitoring android applications on non-rooted devices in the wild," in *Proc. of the 11th ACM Conference on Security & Privacy in Wireless and Mobile Networks*. ACM, 2018, pp. 212–223.

[24] Z. Wang, W. Yan, and T. Oates, "Time series classification from scratch with deep neural networks: A strong baseline," in *Proc. of the 2017 international joint conference on neural networks*. IEEE, 2017, pp. 1578–1585.

[25] M. Willocx, J. Vossaert, V. Raes, and V. Naessens, "Using android devices as mobile extensible hmis," in *2018 Fifth International Conference on Internet of Things: Systems, Management and Security*. IEEE, 2018, pp. 222–227.

[26] D. Wu, Y. Cheng, D. Gao, Y. Li, and R. H. Deng, "Sclib: A practical and lightweight defense against component hijacking in android applications," in *Proc. of the 8th ACM Conference on Data and Application Security and Privacy*, 2018, pp. 299–306.

[27] L. Zhang, Z. Yang, Y. He, M. Li, S. Yang, M. Yang, Y. Zhang, and Z. Qian, "App in the middle: Demystify application virtualization in android and its security threats," *Proc. of the ACM on Measurement and Analysis of Computing Systems*, vol. 3, no. 1, p. 17, 2019.