ThingNet: A Lightweight Real-time Mirai IoT Variants Hunter through CPU Power Fingerprinting

Zhuoran Li and Dan Zhao
Department of Computer Science
Old Dominion University
{zli003,dzhao}@odu.edu

***Abstract*—Internet of Things (IoT) devices have become attractive targets of cyber criminals, whereas attackers have been leveraging these vulnerable devices most notably via the infamous Mirai-based botnets, accounting for nearly 90% of IoT malware attacks in 2020. In this work, we propose a robust, universal and non-invasive Mirai-based malware detection engine employing a compact deep neural network architecture. Our design allows programmatic collection of CPU power footprints with integrated current sensors under various device states, such as idle, service and attack. A lightweight online inference model is deployed in the CPU for on-the-fly classification. Our model is robust against noisy environment with a lucid design of noise reduction function. This work appears to be the first step towards a viable CPU malware detection engine based on power fingerprinting. The extensive simulation study under ARM architecture that is widely used in IoT devices, demonstrates a high detection accuracy of 99.1% at a speed less than 1ms. By analyzing Mirai-based infection under distinguishable phases for power feature extraction, our model has further demonstrated an accuracy of 96.3% on model-unknown variants detection.**

***Index Terms*—Mirai IoT variants detection, power side-channel auditing, lightweight deep learning, noise reduction**

I. Introduction

In the era of hyper-intelligence & digitization, the Internet of Things (IoT) devices play a crucial role in the flourishing development of IoTs. IoT devices, currently 22 billion connected globally, are equipped with diverse CPU architectures and possess unique characteristics such as limited resources, continuous online connection requirement and the lack of security protection. Meanwhile, cyber attacks on IoT devices are accelerating at an unprecedented rate due to the rising deployment in industry, infrastructure, business, healthcare and homes along with the massive shift towards remote-work across the globe during the COVID-19 pandemic. Most current IoT threats are Mirai and its variants that overwhelmed several high-impact targets e.g., Krebs, OVH, Dyn and Github through distributed denial-of-service (DDOS) attacks. Tons of variants were emerged since the original Mirai source code was leaked online in 2016. Over the years, new variants leverage vulnerabilities to hijack IoT devices and harness the power of botnet with millions of infected devices to launch DDOS attacks, e.g., Mozi malware accounting for 89% of the total IoT attacks detected in 2020 as reported by IBM X-Force. According to Netscout Report, 10 million DDOS attacks were observed in 2020 accompanied with huge upsurge of brute-forcing of access credentials and malware targeting IoT devices. As the IoT devices are expected to grow 13% annually to 41 billion in 2025 according to IoT Analytics, there arises a compelling need to protect them from Mirai-based malware attacks.

The emergence and evolution of Mirai malware and its variations and the fragile IoT ecosystem were comprehensively investigated and discussed in [1], [2]. Due to a lack of security design as well as the specific characteristics of IoT devices such as the heterogeneity of processor architecture, IoT malware detection has evolved as a challenging research focus in recent years. While signature-based IoT malware detection methods [3], [4] can provide first-line of defense, they are not effective against unknown or zero-day malware attacks and cannot keep pace with the evolving rate and sophistication of modern malware. In Cisco 2017 Annual Cybersecurity Report, 95% of malware analyzed were not even 24 hours old. Cybercriminals today employ various techniques, e.g., cryptors, code obfuscation, polymorphism to create malware that can constantly change its identifiable features so as to escape signatures-based detection. As reported by the Ponemon Institute, zero-day malware was used in 76% successful attacks in 2018 [5].

Current research focuses on behavior analysis to analyze and evaluate all potential actions that may be performed by a code, e.g., access to any critical or irrelevant files, processes or internal services to detect all malicious or potentially malicious activities. Many recent research has taken advantage of well-known dynamic or static analysis for detecting IoT malware. The dynamic analysis technique [6], [7] monitors executable codes during run-time to detect abnormal behaviors which provides a more complete picture of behavior analysis, however is resource intensive. In contrast, the static approach [8] analyzes static characteristics such as control flow graph, operation codes, and grey-scale images, etc to detect malicious codes without executing them. It generally runs faster but the correctness of data acquisition can easily be affected by the packing technique used. To improve detection performance, recently deep learning has gained importance in malware detection with heterogeneous feature selection which however is computation costly [9], [10].

There is another line of detection research exploring the postulate that malware can be effectively detected by non-intrusively monitoring side-channel data, e.g., vibrations, timing, temperature, power, and electromagnetic, etc [11]–[17]. Recently, an IoT botnet detection method has been developed by modeling the power consumption of an IoT device, and

978-3-9819263-6-1/DATE22/©2022 EDAA

classifying the malicious behaviors with a convolution neural network deep learning model [18]. Unfortunately this approach neither tackles the problem of noisy power signals nor considers consecutive mixed device activities without clear boundaries. Further, it relies on an expensive off-the-shelf HV Power Monitor to monitor power consumption. Collection and storage of power datasets and execution of classification offline on a computer involve additional detection overhead and latency. In this paper, we build a lightweight malware detection engine, dubbed IoT Malware Hunter, deployed in an ARM processor that entails the integrated current sensors to extract the malicious power signals from ambient noise for on-the-fly detection. Our contribution can be summarized as follows:

- We develop a *lightweight* deep learning model namely *ThingNet* tailed for resource constrained embedded processors, e.g., ARM Cortex-A series processors widely used for IoT devices, by implementing depthwise separable convolutions to greatly compress model operations while maintaining detection accuracy. An elastic contractible soft thresholding is further integrated as nonlinear transformation layers to eliminate non-essential features to achieve denoising.
- We implement the CPU detection engine, namely *IoT Malware Hunter* on ODROID-XU3 with the Samsung Exynos5422 Octa CPU to command for data streaming, preprocessing and classification in *real time*. Two integrated Texas Instruments INA231 current/voltage sensors instantaneously monitor the power consumption of the Big A15 cores cluster and Little A7 cores cluster respectively to identify malware attack at early infection stages. The system design is fine tuned for best tradeoff between model size, speed and accuracy.
- Our detection engine is *robust* even in the real open world noisy environment where an IoT device runs various applications and services when getting infected by a malware. Our simulation study demonstrates that ThingNet can achieve a high detection accuracy of up to 99.1% at $0.4ms$.

The rest of paper is organized as follows. Sec. II takes a system overview of the IoT Malware Hunter with both hardware and software modules. It further presents our study of the Mirai infection process and discusses the threat model. Sec. III describes the design of a lightweight deep-learning model ThingNet for real-time malware infection detection under noisy environment. The extensive simulation study is performed in Sec. IV and finally Sec. V concludes this work.

II. System Design of IoT Malware Hunter

IoT Malware Hunter is designed to detect malware attacks at the early infection stages to prove that such malware actions leave detectable fingerprints on power side-channel data. We will analyze malware infection activities and its correlation with power variations to further experiment with deep learning techniques for efficient power feature extraction and infection detection. Moreover, we target at Mirai malware [1] as Mirai variants continue to evolve to support malware propagation and infection across different platforms and architectures.

A. System Overview

The IoT Malware Hunter is an integral design of hardware platform and software stack. As shown in Fig. 1, the detection engine hardware is built by reusing the on-device embedded processor or microcontroller (e.g., Samsung Exynos5422 Octa CPU on ODROID-XU3) and the built-in power sensor(s) (e.g., TI INA231 current/voltage sensors primarily used for power management) connected to the CPU via I2C bus(es). The sensor(s) constantly monitors the CPU power consumption and the CPU live streams power data, preprocesses data segments, detects infection activities and instantly alerts the system of malware attacks. The challenge is how to accurately detect such infections by identifying distinguishable power fingerprints on a resource-constrained IoT device.

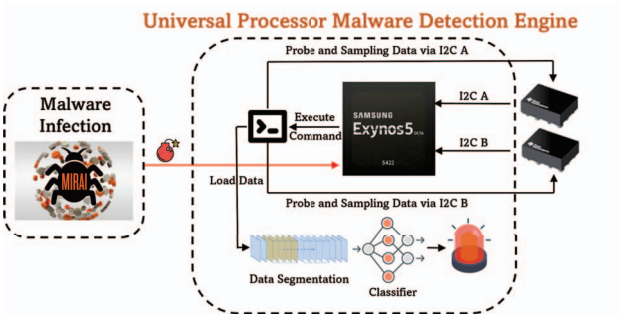

Fig. 1: System overview

The detection engine software module mainly includes three phases (1) instant power data collecting via current sensor(s), (2) data segmentation with adaptive sliding window with overlapping approach, and (3) power feature extraction and classification. As the entire process is dominated by power auditing involving power data collection (in phase 1) and preprocessing (in phase 2), we will interleave the processes from multiple sensors (for heterogeneous multicore processors) so that ThingNet can simultaneously detect up to $\frac{t_{Col}+t_{Seg}}{t_{Inf}}$ sensor datasets (where t_{Col}, t_{Seg} and t_{Inf} denote the average time needed for data collection, segmentation and classification respectively). To enhance power feature extraction, we fine tune the current sensor sampling rate, and sliding window size and overlapping percentage for desired datasets collection. In order to build the online inference model, we first conduct offline training to determine the architecture and parameters of the deep learning model which will be discussed in Sec. III.

B. Threat Model

The IoT Malware Hunter is designed based on the postulate that when device operations are changed by malicious activities, they leave physical traces that can be distinguishably identified via side channels. For example, we are able to observe the Mirai malware infection process via power side-channel auditing. As illustrated in Fig. 2, the power waveform in three red boxes correspond to a sequence of malicious activities during the Mirai infection process (a) Telnet scanning (b) Reporting IP (c) Loading bot in Fig. 3. As we can see these three waveform

are different and perceptible from those under device normal operations.

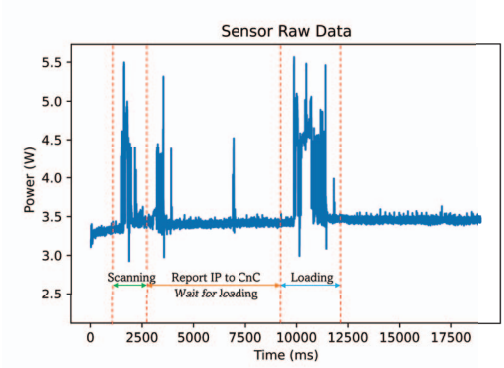

Fig. 2: Power consumption during Mirai infection

In this work we focus on the infection of Mirai family on ARM-based hardware architecture (e.g., ARM Cortex A series CPU) widely used for IoT devices. For instance in Fig. 3, each Mirai infected device scans the Internet for open Telnet ports 23 or 2323 (step 2). Once a vulnerable victim is found, the malware attempts to perform a brute-force login using a list of 62 known default credentials of BusyBox-based IoT devices. If the login attempt is successful, the device IP and credentials are sent to a centralized ScanListen service (step 3) which are further used by the bot-load service that subsequently loads and executes the bot on the new victim (step 5). The malware binary is then removed and runs only in memory to avoid detection. Each bot will repeat this process to propagate and infect more vulnerable devices. Such self-replication results in an exponential growth in the botnet size up to 500 brute results per second at peak as reported by Virus Bulletin. We aim at the detection of malicious behaviors when devices undergo the above three steps in the Mirai infection process.

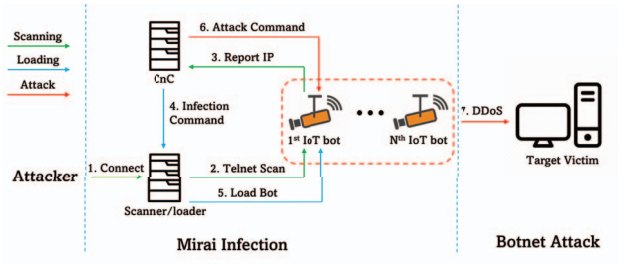

Fig. 3: Mirai infection process

C. Design Optimization of Power Fingerprinting

The power auditor implements the functions of data collection and preprocessing. The raw power data is sampled and instantly collected from the current sensor(s) via I2C bus(es). The sampling rate plays an important role in detection performance as the false alarm rate and miss detection rate depend upon the number of samples. We will explore the effect of sampling rate on detection performance and investigate the minimum sampling rate that maintains high detection performance in terms of accuracy and precision. For effective feature extraction and classification, the continuous power data stream is divided into discrete segments by applying an adaptive window segmentation and feature construction protocol. The sliding-window based segmentation methods have been widely used for dividing the activity data steam [19], [20]. The challenge is how to optimize window sizing and overlapping to maximize detection performance. We will exploit the impact of window sizing with overlapping on the detection of infection activities and automatically adapt window sizes for different activities.

III. Design of ThingNet

Aiming at the design of a lightweight deep learning model that can accurately detect the infection activities on-the-fly under a noisy environment, we propose ThingNet tailed for resource constrained IoT devices. ThingNet integrates the idea of depthwise separable convolutions [21] to develop a compact architecture that greatly reduces network parameters (e.g., the number of weights and operations) while maintaining detection accuracy. The key design lies in twofold (1) using a series of smaller filters instead of a large filter to reduce the total number of weights; and (2) applying these smaller filters in sequence to achieve the same overall effective receptive field of a classical convolution layer. In consideration of power data collection under a noisy environment where the embedded processor runs various applications and services while under attack, signal denoising is essential to achieve reliable detection performance. We thus develop a new idea to seamlessly integrate the signal denoising functionality in a compact deep learning architecture to enhance detection performance under the real-world noisy environment. Consequently, an elastic contractible soft thresholding method is further developed and integrated in ThingNet as nonlinear transformation layers to eliminate non-essential features, i.e., noises. Derived from the widely used denoising approach of soft thresholding [22], our approach adaptively adjusts the threshold via an content aware mechanism that compresses a global view of each channel into a single value while elastically weighting each feature map.

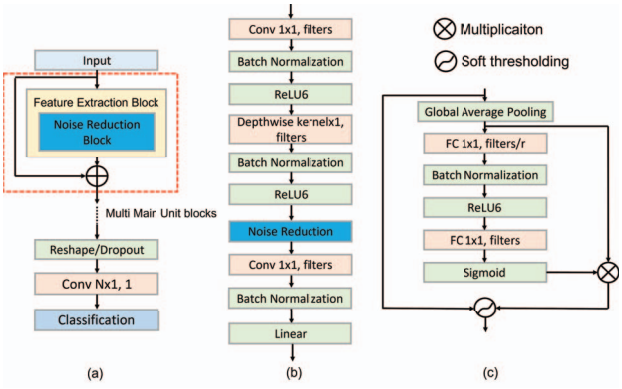

Fig. 4: ThingNet Design (a) main structure (b) feature extraction block (c) noise reduction block

In this section, we describe our proposed architecture of ThingNet that mainly consists of two building blocks as shown

in Fig. 4(a). The feature extraction block as shown in Fig. 4(b) has an expansion layer, $1D$ depthwise convolution layer, a noise reduction block and a pointwise convolution layer plus batch normalization and activation functions after each layer respectively. The expansion layer is designed by using 1×1 convolution kernel to increase the feature information that can be extracted from the input data. The channel number is increased by multiple times in correspondence to the number of smaller filters used. This process is an inverted way of traditional convolution neural network (CNN) as it gradually reduces the numbers of filters used by deeper layers. A depthwise convolution layer is followed after the expansion layer to greatly reduce computation needed for the model. As a result, ThingNet only requires at most 13% calculations as of standard CNN operations. We repeat the main unit blocks a couple of times with the same structure but different configuration of parameters to extract more complicated features in the deep layers as shown in Table I.

Algorithm 1 ThingNet Main Unit Blocks

Input: $x : 1 \times W \leftarrow$ 1-D power data with a length of W
Output: $y : 1 \times W \times C \leftarrow$ denoised feature map
Parameters:
$n \leftarrow$ numbers of main unit blocks, $\kappa \leftarrow$ scaling factor
$C_i \leftarrow$ numbers of filters in expansion layer
$K_n \leftarrow$ different kernal sizes, $r \leftarrow$ reduction ratio
Procedure:
i=1
while $i \leq n$ **do**
  $x : 1 \times W_i \xrightarrow{Conv1 \times 1, C_i} x : 1 \times W_i \times C_i$ ▷ Feature expansion
  $x : 1 \times W_i \times C_i \xrightarrow{Conv1 \times K_i, C_i} x : 1 \times W_i^{'} \times C_i$ ▷ Depthwise convolution
  $x : 1 \times W_i^{'} \times C_i \xrightarrow{GAP} x_{avg} : 1 \times 1 \times C_i$ ▷ Squeeze
  $x_{avg} : 1 \times 1 \times C_i \xrightarrow{FC} \kappa : 1 \times 1 \times C_i/r_i$
  $\kappa : 1 \times 1 \times C_i/r_i \xrightarrow{FC} \kappa : 1 \times 1 \times C_i$ ▷ Excitation
  $\alpha = \kappa \otimes x_{avg}$ ▷ Soft thresholding value
  switch x **do** ▷ Delete noise
    case $x > \alpha \rightarrow y = x - \alpha$
    case $-\alpha < x < \alpha \rightarrow y = 0$
    case $x < -\alpha \rightarrow y = x + \alpha$
  $y : 1 \times W_i \times C_i \xrightarrow{Conv1 \times K_i^{''}, C_i^{'}} y : 1 \times W_i^{''} \times C_i^{'}$ ▷ Pointwise convolution
  i++
end while

To achieve denoising, a noise reduction block as shown in Fig. 4(c) is integrated inside the feature extraction block to further improve detection performance. It uses a threshold value to filter unrelated information while increasing the weight of important features. The key design of noise reduction block is to apply the squeeze-and-excitation [23] functionality to automatically train a soft threshold value [24] that could be used as a noise filter for each input data instance. Our soft thresholding function can effectively eliminate the non-essential feature beyond the threshold interval. As a result, the Mirai infection features remain while the noise features (i.e., running applications and services) are trimmed.

Specifically, the noise reduction block has one global average pooling (GAP) layer, two fully connected (FC) layer, one batch normalization layer and a soft thresholding layer as shown in Fig. 4(c) and in Algorithm 1. The GAP layer is added to calculate a mean value from each channel of the feature map. Then follows, ThingNet trains and learns feature scaling by using two fully connected layers. Aiming at squeezing unimportant features while exciting important features, we assume that the data before GAP has a shape of $1 \times W \times C$, GAP calculates an average value over C channels in order to apply the scale to each channel eventually. The generated feature map after GAP can be denoted as $1 \times 1 \times C$ (indicating the process of squeezing). Note that the scale numbers from GAP cannot fully represent entire feature maps because they are generated from each single channel. The first FC layer thus compresses C channels into C/r channels so as to use the second FC layer to extract the scales correlated with individual channels (indicating the process of excitation). Here, the reduction ratio factor r can be determined via experiments. A sigmoid activation layer is used to ensure that scale factor is between [0,1]. The soft threshold value is calculated by multiplication of average value calculated from the GAP layer and the scale factor extracted after sigmoid activation function. The soft thresholding value is then applied to filter noises after the depthwise convolution layer. By implementing the noise reduction block with a squeeze-and-excitation mechanism, ThingNet can detect more important features such that the Mirai infection activities will be excited and extracted.

IV. Experiments

We implement a prototype Mirai Variants Hunter system to run experiments and evaluate the performance of ThingNet.

A. Power Dataset Collection

Hardkernel Odroid-XU3 installed with Ubuntu 16.04 is used as our experiment environment to implement the prototype system of Mirai Variants Hunter. Odroid-XU3 is integrated with ARM big.Little processor SoC to run applications and services along with four power sensors (i.e., Texas Instrument INA231) for efficient power management of core clusters. These sensors are reused in our system to collect the power data of the ARM processor at a sampling rate ranging between 50-1k Hz. A Dell Precision 5520 Laptop installed with Ubuntu 18.04 acts as the attack machine to launch the Mirai family botnet attack (as described in Fig. 3) on the victim IoT device of Odroid-XU3. The lightweight inference model ThingNet runs on Odroid-XU3 to real-time detect botnet attacks at the early malware infection phase. The experimental setup is illustrated in Fig. 5.

The power data is acquired under four most representative states of an IoT device, i.e., idle, under Mirai infection, under normal services, and service device under Mirai infection. The detailed description of each type of data is given below.

- Idle: The victim machine is not running any apps for most of the time.

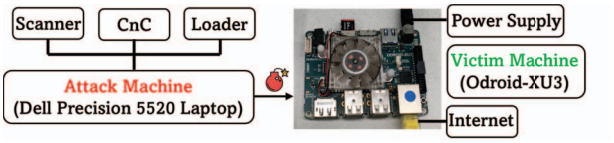

Fig. 5: Experimental setup

Input (1D), Channels	Operator	Filters	Kernel size	Strides
300×1, 1	Conv1D	8	8	2
150×1, 8	Main Block	32	4	2
75×1, 32	Main Block	96	16	1
75×1, 96	Main Block	144	16	1
75×1, 144	Conv1D	24	1	2
38×1, 24	GAP	-	-	-
1×1, 24	Conv1D	4	1	1

TABLE I: Model structure parameters

- Under Mirai infection: The device experiences the infection process includes scanning, IP reporting and loading.
- Under normal service: Various applications such as Minecraft gaming and OpenHAB smart home system run on the victim.
- Service device under Mirai infection and noisy apps: Attacker randomly launches either Mirai scanning or loading under any app of Minecraft and OpenHAB to demo a noisy environment.

About 3000 power data samples are collected under each state, each sample with a length of 3s-5s.

B. Classifier Optimization

A compact offline training model is developed based on the design in Sec. III to effectively extract power feature and classify four different device states. The model is trained with 5-fold cross-validation and the parameters are summarized in Table I. In consequence, the TensorFlow Lite converter [25] is used to convert the trained compact model into a lightweight inference model that can run on Odroid-XU3 for real-time detection under noisy environment. The ThingNet performance is evaluated in terms of accuracy, precision, recall and F1-score.

1) Power Feature Extraction Optimization: The impact of power data collection and preprocessing on feature extraction is evaluated taken into the consideration of sampling rate, sliding window size and overlapping rate. The parameters are fine tuned to achieve the best model performance. Note that all the experiments are conducted with a fixed two out of three variables:e.g. when running experiments on the effect of sampling rate, we set sliding window size as $1.5s$ and overlapping rate as 75%. As we can see, the higher the sampling rate, the better the performance achieves. For example, the highest accuracy of 97.8% is achieved at the largest sampling rate of $1KHz$. A 6%-7% performance reduction is observed under the sampling rate of $100Hz$. It is due to the missing representative patterns in the collected input dataset. Meanwhile, the performance nearly saturates at a sliding window size of $1.5s$. When the sliding window size is reduced to $500ms$, the performance is reduce by 6%-7% as well. In general, the performance improves with the increase of overlapping percentage. The 75% overlapping is our best choice as it reaches the highest performance and the lowest false positive rate. However we can trade-off between the overlapping resources and the desired performance. For instance, though the accuracy and false positive rate may be reduced by 2%-3%, 50% of overlapping resources can be saved.

2) Model-Unknown Variants Detection: Satori, Okiru and Masuta have evolved after Mirai and inherit Mirai family's functionality. Thus similar side-channel power patterns can be observed when running the same Linux commands as Mirai. We develop our model against these Mirai variants to demonstrate its capability of detecting unknown variants as summarized in Table III. Two scenarios are defined by considering either Mirai infection as one class or splitting the infection process power data into Mirai scanning and Mirai loading as two classes. As shown in Table III, when the Mirai infection process is segmented into two separate events, the accuracy is improved by 10%-12%. In other words, a fine-grained analysis of the correlation between the distinguishable power patterns and a sequence of malware execution behaviors results in a more robust model for Mirai variants detection than the coarse-grained approach in [18] that segmented the entire infection process as a single event. As a result, our model can effectively identify common functions among Mirai variants while filtering out unrelated features belonging to the original Mirai that however cannot help differentiate unknown variants. We claim that incorporating fine-grained analysis of power data into model development may potentially improve detection performance on unknown malware and zero-day attacks.

C. Malware Hunter System Performance Evaluation

We use Tensorflow Lite Converter [25] to deploy ThingNet inference model on device. The deployment takes about $1MB$ storage depending on the type of device. The prototype system performance of malware hunter is further evaluated in terms of model size, detection speed, and runtime memory usage. A 8-layer CNN model as developed in [18] is also implemented as a base of comparison. The power data is collected and preprocessed with $1KHz$ sampling rate, $1.5s$ sliding window size and 75% overlapping. The detection speed is calculated for a duration of raw data preprocessing and classification as shown in Fig. 6. The memory usage is estimated by inserting two anchors *tracemalloc.start()* and *tracemalloc.end()* at *Start* and *Stop* respectively. The *tracemalloc* function will take a snapshot of memory allocated between the two time anchors.

The comparison result of 2-class (i.e., device infection with noisy apps and no infection) and 4-class (i.e., idle, Mirai infection, normal service, and service device under Mirai infection and noisy apps as defined in Sec. IV-A) classifiers is given in Table IV. As ThingNet is designed for reliable malware infection detection in noisy power data, it outperforms classical CNN classifier with 4% and 13% improvement for 2-class and 4-class respectively. Designed with the compact architecture based on depthwise separable convolution structure, ThingNet model size is reduced by 41% and its detection speed is about 6 times faster and its runtime memory usage is reduced by half as compared to the base. In a nutshell, the compact architecture

Metrices	**Sampling Rate** (Hz)				**Sliding Window Size** (ms)			**Overlapping** (%)			
	100	200	500	1000	500	1500	3000	0	25	50	75
Accuracy (%)	90.2	**96.5**	97.2	97.8	90.2	**96.5**	97.2	90.6	93.4	92.9	**96.5**
Precision (%)	88.69	**96.42**	96.85	97.64	89.26	**96.42**	96.47	91.3	95.95	93.8	**96.42**
Recall (%)	91.91	**96.61**	97.6	98.03	91.3	**96.61**	98.0	90.3	91.15	92.14	**96.61**
F1-Score (%)	90.27	**96.5**	97.23	97.83	90.27	**96.5**	97.24	90.8	93.49	92.96	**96.5**

TABLE II: Impact of Sampling Rate, Sliding Window Size and Overlapping Percentage

Scenarios (Accuracy%)	Input Data	Model Unknown Variants		
	Mirai	**Satori**	**Okiru**	**Masuta**
2 class (Mirai infection, No infection)	99.1	87.6	81.4	86.5
3 class (Mirai scanning, Mirai loading, No infection)	96.5	96.3	92.5	88.6

TABLE III: Detection on Mirai variants

design along with the denoising approach makes ThingNet a viable solution of CPU IoT botnet detection engine.

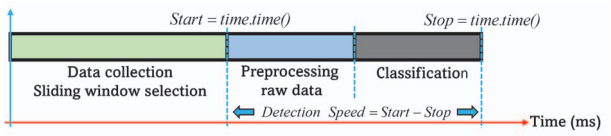

Fig. 6: Illustration of detection process timeline

Metrices	**4 class (Mirai, Idle, Service under Mirai&Random App, Normal Service)**		**2 class (Infected with Mirai, No Infection)**	
	8-Layers CNN	ThingNet	8-Layers CNN	ThingNet
Accuracy (%)	83.2	96.2	95.3	99.1
Model Size (kB)	876	360	878	361
Detection Speed (ms)	18.306	3.247	18.279	3.247
Mem Usage (kB)	3.845	1.841	3.844	1.727

TABLE IV: System performance comparison

V. Conclusion

In this paper, we have designed a compact malware detection engine, namely ThingNet running on the ARM processor by auditing side-channel power data under noisy environment. The experiments have demonstrate that ThingNet can effectively identify abnormal power behaviors during malware infection, achieving a high detection accuracy of 99.1% with false positive rate of 0.21%. With fine-grained power modeling, ThingNet has further demonstrated its robustness in detecting unknown variants at an accuracy of 96.3%. ThingNet is deemed a lightweight real-time detection engine with about $3ms$ detection speed and $1.8KB$ runtime memory usage.

References

[1] M. Antonakakis and et. al, "Understanding the mirai botnet," in *26th USENIX security symposium*, 2017, pp. 1093–1110.

[2] Z. Ling and et. al, "New variants of mirai and analysis," in *Encyclopedia of Wireless Networks*. Springer International Publishing, 2020, pp. 1–8.

[3] M. Abbas and T. Srikanthan, "Low-complexity signature-based malware detection for iot devices," in *International Conference on Applications and Techniques in Information Security*, 06 2017, pp. 181–189.

[4] M. Alhanahnah and et. al, "Efficient signature generation for classifying cross-architecture iot malware," in *2018 IEEE Conference on Communications and Network Security (CNS)*, 2018, pp. 1–9.

[5] "The 2018 state of endpoint security risk," https://www.ponemon.org/news-updates/news-press-releases/news/the-2018-state-of-endpoint-security-risk.html.

[6] J. Jeon, J. H. Park, and Y.-S. Jeong, "Dynamic analysis for iot malware detection with convolution neural network model," *IEEE Access*, vol. 8, pp. 96 899–96 911, 2020.

[7] R. Kumar and et. al, "Iotmalware: Android iot malware detection based on deep neural network and blockchain technology," 2021.

[8] A. Azmoodeh, A. Dehghantanha, and K.-K. R. Choo, "Robust malware detection for internet of (battlefield) things devices using deep eigenspace learning," *IEEE Transactions on Sustainable Computing*, vol. 4, no. 1, pp. 88–95, 2019.

[9] C.-W. Tien and et. al, "Machine learning framework to analyze iot malware using elf and opcode features," *ACM Digital Threats: Research and Practice*, vol. 1, no. 1, Mar 2020.

[10] D. Vasan and et. al, "Mthael: Cross-architecture iot malware detection based on neural network advanced ensemble learning," *IEEE Transactions on Computers*, vol. 69, no. 11, pp. 1654–1667, 2020.

[11] M. Guri, "Air-viber: Exfiltrating data from air-gapped computers via covert surface vibrations," *CoRR*, vol. abs/2004.06195, 2020.

[12] S. J. Stone, M. A. Temple, and R. O. Baldwin, "Detecting anomalous programmable logic controller behavior using rf-based hilbert transform features and a correlation-based verification process," *Int. J. Crit. Infrastruct. Prot.*, vol. 9, no. C, p. 41–51, Jun. 2015.

[13] S. S. Clark and et. al, "Wattsupdoc: Power side channels to nonintrusively discover untargeted malware on embedded medical devices," in *2013 USENIX Workshop on Health Information Technologies (HealthTech 13)*, Washington, D.C., Aug. 2013.

[14] J. Hoffmann, S. Neumann, and T. Holz, "Mobile malware detection based on energy fingerprints - a dead end?" in *Int. Workshop on RAID*, 2013, pp. 348–368.

[15] Y. Liu and et. al, "On code execution tracking via power side-channel," in *Proc. of the ACM SIGSAC Conference on Computer and Communications Security*, 2016, p. 1019–1031.

[16] S. Wei, A. Aysu, M. Orshansky, A. Gerstlauer, and M. Tiwari, "Using power-anomalies to counter evasive micro-architectural attacks in embedded systems," in *2019 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 2019, pp. 111–120.

[17] A. Nazari, N. Sehatbakhsh, M. Alam, A. Zajic, and M. Prvulovic, "Eddie: Em-based detection of deviations in program execution," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, 2017, pp. 333–346.

[18] W. Jung and et. al, "Iot botnet detection via power consumption modeling," *Smart Health*, vol. 15, p. 100103, 2020.

[19] B. Fida and et. al, "Varying behavior of different window sizes on the classification of static and dynamic physical activities from a single accelerometer," *Medical Engineering & Physics*, vol. 37, no. 7, p. 705—711, July 2015.

[20] J. Wan and et. al, "Time-bounded activity recognition for ambient assisted living," *IEEE Transactions on Emerging Topics in Computing*, vol. 9, no. 1, pp. 471–483, 2021.

[21] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1251–1258.

[22] D. L. Donoho, "De-noising by soft-thresholding," *IEEE transactions on information theory*, vol. 41, no. 3, pp. 613–627, 1995.

[23] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 7132–7141.

[24] M. Zhao and et. al, "Deep residual shrinkage networks for fault diagnosis," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 7, pp. 4681–4690, 2019.

[25] "Tensorflow lite converter," https://www.tensorflow.org/lite/convert.