Summer 2024

Privacy-Preserving Deep Learning Framework for IoT Malware Detection

Sabbir Ahmed Khan
Old Dominion University, sabbir042@gmail.com

PRIVACY-PRESERVING DEEP LEARNING FRAMEWORK FOR IOT MALWARE DETECTION

by

Sabbir Ahmed Khan
B.S. March 2009, Bangladesh University of Engineering and Technology (BUET), Bangladesh
M.S. December 2022, Old Dominion University

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

DOCTOR OF PHILOSOPHY

COMPUTER SCIENCE

OLD DOMINION UNIVERSITY
August 2024

ABSTRACT

PRIVACY-PRESERVING DEEP LEARNING FRAMEWORK FOR IOT MALWARE DETECTION

Sabbir Ahmed Khan
Old Dominion University, 2024
Co-Directors: Dr. Danella Zhao
Dr. Ravi Mukkamala

Cyberattacks on IoT devices are accelerating at an unprecedented rate, largely driven by IoT malware activities. The IoT malware attacks typically comprise three stages: intrusion, infection, and monetization. Existing IoT malware detection methods fail to identify malicious activities at the intrusion and infection stages and thus cannot stop potential attacks timely. In our research, we have leveraged power side-channel information as input to our deep learning model to identify malware at early stages of intrusion on IoT devices. But, deploying a resource-intensive deep learning model on highly resource-constrained IoT devices is a significant challenge. Consequently, utilizing a Machine Learning as a Service (MLaaS) engine to offload computation tasks to edge servers in the cloud becomes an attractive solution. However, edge computing introduces significant privacy concerns since client data from IoT devices is sensitive, and the model parameters at the edge server are regarded as proprietary information. Therefore, we propose three privacy-preserved deep learning frameworks to monitor side-channel power consumption in real-time and identify its correlation to various malware infection activities without leaking client or server information. Our first framework, DeepShield, is a secure inference-based IoT malware detection system characterized by a novel hybrid cryptographic protocol. This protocol offloads most computation to the edge and enables secret-sharing collaboration between the client and edge server. It takes the most expensive computation of homomorphic operations offline, lightening online secure interaction. However, its detection strategy must catch up with the rapid pace of malware evolution. Hence, we introduce our second framework, BoTShield, a novel privacy-preserved online training method capable of detecting malware variants. We use a combination of homomorphic encryption, secret sharing, and differential privacy approach to preserve the privacy of BoTShield. Though BoTShield represents an advancement over DeepShield, it isn't fully equipped to detect zero-day malware attacks. Thus, we introduce MalwareShield, a privacy-preserved federated learning framework based on a novel differential privacy approach equipped with an encoder-based unsupervised model to detect zero-day malware attacks. Moreover, MalwareShiedl reduces the amount of data communication between the client and the server. Our empirical experiments demonstrate that these frameworks enable secure, accurate, real-time, and scalable malware detection.

I dedicate this dissertation to my beloved family members, whose unwavering love, support, and sacrifices have been the cornerstone of my academic journey. Additionally, I dedicate this work to all those who endure the profound and enduring impacts of war. May their resilience inspire us to strive for a world of peace, compassion, and understanding.

ACKNOWLEDGMENTS

I am sincerely grateful to Dr. Danella Zhao for her invaluable guidance, insights, and unwavering support throughout my Ph.D. journey. Her expertise has been instrumental in shaping my academic path and personal growth.

I extend my gratitude to Dr. Ravi Mukkamala, my co-advisor; his encouragement has instilled in me the determination and passion to tackle future challenges confidently. Additionally, I extend my gratitude to committee members Dr. Stephan Olariu and Dr. Chunsheng Xin for their insightful comments and steadfast support. I also wish to express appreciation to Dr. Michele Weigle, the CS graduate program director, for her invaluable assistance and guidance throughout my Ph.D. journey.

I would like to express my gratitude to Zhuoran Li and Woosub Jung for providing the power dataset. Special thanks go to Zhuoran Li and Dr. Danella Zhao for their invaluable assistance with writing this dissertation. I am also grateful to Dr. Gang Zhou, Dr. Chunsheng Xin and Qiao Zhang for their contributions during the initial stages of this work.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

The rising popularity of Internet of Things (IoT) technology has made IoT devices a favorite target for cybercriminals due to its heavy resource constraints, constant Internet connection, and lack of built-in security [2], [6]. IoT devices have grown exponentially in recent years, with a projected 75 billion units by 2025, a significant rise from 35 billion units in 2021 [51]. However, the rapid increase in IoT devices introduces unexpected risks and new threats. According to the 2023 report by siliconANGLE [71], the infamous Mirai family of malware continues to evolve, posing a significant menace to the IoT landscape through its numerous variants. Cyberattacks on IoT devices are accelerating at an unprecedented rate, primarily driven by IoT botnet activities [3]. For example, in 2016, a massive DDoS attack knocked down the DNS service Dyn and made a catastrophic East Coast Internet outage, which Mirai Botnet caused. Since then, new variants are evolving and exploring a variety of vulnerabilities in unsecured IoT devices. For instance, IBM observed that a Mirai variant, Mozi malware, has accounted for 89% of the total IoT attacks detected for 2020. It has become imperative to detect IoT botnet attacks promptly to expedite the alerting and disconnection of compromised IoT devices from the Internet, which helps stop and prevent ever-more sophisticated attacks.

IoT botnet attacks such as Mirai are typically performed through four distinct operations, e.g., propagation, infection, command and control (C&C) communication, and attack execution [41]. Various IoT botnet detection techniques have been developed to identify distinguishable footprints at different operational steps. From the data perspective, some approaches focus on malware source/binary analysis [24], [72] while others use host or network-based behavior features [9], [57], [66]. Meanwhile, the dynamic analysis method has been applied to network-level traffic analysis to determine the characteristics for building network intrusion detection systems [31], [55]. Moreover, some other anomaly detection methods have been proposed in [28], [54] by extracting C&C network behavior. However, these approaches mainly focus on behavior analysis in later steps of the IoT botnet attack process, i.e., C&C communication and attack execution. To minimize the massive damage caused by IoT botnet attacks as early as possible, it is essential to identify malicious activities at early steps, i.e., propagation and infection.

To this end, several recent research explore the correlation between malicious actions and physical responses at early steps, e.g., power observation via side channels [13], [19], [36]. This inspired us to analyze the infection activities of Mirai and its variants, which are among the main threats to IoT devices, particularly when these malware execute brute-force scanning (to discover credentials) and loading activities (to gain access and download and execute malware) on the victim IoT device. Specifically, we find that device power consumption leaks distinguishable information about Linux command execution to perform Mirai family functions compared with the normal operations when the IoT device is idle or in service. The question is how to accurately identify such runtime malicious behaviors by instantaneously auditing the power consumption of the IoT device during malware execution. Few works have been done to address such an imperative problem.

1.1 PROBLEM

Detecting malicious activities in the early stages, particularly during propagation and infection, is crucial to mitigate the extensive damage inflicted by IoT botnet attacks as swiftly as possible. For the real-time detection of IoT botnets, we've proposed several deep-learning models that utilize input from embedded current sensors to extract power features from malicious and legitimate activities. While deep learning is rapidly advancing and proven to be an effective tool for run-time behavior dynamic analysis [34], [64], [69], it is challenging to deploy resource-hungry deep learning model on heavily resource-limited IoT devices. Therefore, it becomes appealing to use a Machine Learning as a Service (MLaaS) engine to outsource computation tasks to edge servers in the cloud, which supports live streaming of power data from the IoT devices to the server and on-the-fly botnet detection at the server which has a well-trained deep learning model, as such we can expedite the alerting and instantly prevent the botnet from propagation and potential massive attacks. However, edge computing raises serious privacy issues [30], [84] as client data from IoT devices is very privacy-sensitive, and model parameters at the edge server are considered proprietary information. In this work, we develop a smart auditor that mediates between resource-constrained IoT devices and edge server with a novel design of privacy-preserving interactions during MLaaS, aiming to instantly monitor side-channel power consumption and identify its correlation to various botnet infection activities without leaking client and server information.

1.1.1 Approach 1: DeepShield

Advanced cryptographic primitives, e.g., homomorphic encryption, have been introduced [21], [37] to run CNN over encrypted data with high accuracy securely. However, heavy online execution of cryptographic operations demands substantial hardware resources, which induce impractical computation and communication overhead (e.g., about 300-second secure inference in CryptoNets [21]. Therefore, this type of resource-demanding computation can not be directly applied to the real-time detection of IoT Botnet attacks. On the other hand, computation performance can be primarily optimized by lifting off the heavy cryptographic operations offline [18], [48], [63].

Unfortunately, using privacy-preserving primitives such as garbled circuits [83] for online execution still involves substantial computing resources. It has evolved as a timely challenge, among the current works, to facilitate secure and efficient CNN online inference for the Internet of Things and intelligent infrastructure. Our goal is thus to develop an extremely lightweight framework for privacy-preserving CNN to achieve feature extraction and classification for real-time IoT botnet detection under edge computing. We propose a novel lightweight privacy-preserving inference framework, DeepShield, that can detect real-time malware attacks.

1.1.2 Approach 2: BoTShield

The above method is an inference model, and the training is conducted offline. Later, the offline training model parameters are imported into the inference model to detect the malware. However, owing to the rapid evolution of malware, traditional offline-trained on-device malware detection methods must catch up with the swift evolution of the current IoT environment. Recent reports in [4] have revealed that 560,000 new malware are detected daily, equating to more than six new malware variants generated every second. In 2021, ransomware attacks occurred every 11 seconds, and this frequency is projected to increase to every 2 seconds by 2031 [15]. Hackers extorted 1.1 billion in 2023, marking a twofold increase compared to 2022. Therefore, to effectively detect IoT malware, online training with real-time collected malware data can provide considerable advantages to protect IoT devices. We propose BoTShield, a malware detection system that utilizes a novel "secret sharing with two-party computation" approach to facilitate secure online training. Specifically, this approach enables the rapid adaptation to new malware patterns upon initial exposure. It is further fortified with multiple secure protocols to ensure the security of online training against malware threats. The problem is modeled as follows: a set of M data owners send their input data to the server. The servers run an interactive protocol with clients to train a neural network over the data to produce a trained model that can be used for inference. The privacy requirement is that no individual party or server learns any information about any other party's data; thus, our model is designed in a two-party semi-honest setting. In this model, we adopt a hybrid cryptographic approach different from other state-of-the-art work [56], [78]. Most of the state-of-the-art works take a long time to train the model. Therefore, we propose this hybrid approach that computes the heavy cryptographic operation during the offline phase by using a secret sharing protocol, thus significantly reducing the computation time of our model and achieving real-time malware detection during online inference. Another unique feature of this model is to enable activation without approximation because the client now computes it in plaintext form. The client now computes it, thus ensuring that it is free of accuracy loss and the desired stability in training.

1.1.3 Approach 3: MalwareShield

A zero-day attack on IoT devices exploits vulnerabilities in software or firmware that are not yet known to developers or manufacturers. Malicious actors use these vulnerabilities to compromise IoT devices, gain unauthorized network access, steal data, or launch larger-scale attacks. The impact can be severe, especially if the compromised devices are part of critical infrastructure or industrial systems. Such devices can create botnets for distributed denial-of-service (DDoS) attacks, infiltrate corporate networks for espionage or data theft, or disrupt essential services.

Traditional security solutions, like BoTShield discussed in Chapter 5, rely on supervised learning and require labeled malware data for training. This dependency limits their ability to detect previously unseen malware, particularly zero-day attacks. To overcome this limitation, we propose MalwareShield, a privacy-preserving federated learning framework that employs our novel differential privacy approach to detect new malware variants, including zero-day exploits. Unlike BoTShield, MalwareShield is trained exclusively on benign data, eliminating the need for labeled attack data.

We incorporate federated learning into the framework because traditional centralized deep learning (CDL) methods do not ensure the privacy and security of IoT devices, as they involve transmitting data from all participating devices to a central cloud server. By combining edge computing with deep learning, MalwareShield brings intelligence closer to the data generation points, addressing issues such as data privacy, high communication costs, extensive memory requirements, long training times, and high latency. MalwareShield facilitates collaborative deep learning in distributed IoT devices without sharing private data with the central cloud server, thus ensuring data privacy, minimizing communication costs, and reducing training time. By leveraging deep learning techniques, MalwareShield can detect abnormal patterns indicative of malware with high accuracy, even for previously unseen attack data.

1.2 CONTRIBUTIONS

The main contribution of our work include:

- We propose a novel lightweight privacy-preserving inference framework, DeepShield, to securely offload over 90% of CNN inference operations to the edge and enable efficient secret-sharing collaborative computation between smart auditor and edge server with small overhead communication overhead. Since most of the calculation is delegated to the edge server, it facilitates low-cost and scalable implementation of a smart auditor to monitor an extensive collection of IoT devices simultaneously.

- Due to the rapid evolution of malware, traditional offline-trained on-device malware detection methods cannot keep up with the swift changes in the current IoT environment. Therefore, we propose BoTShield, a novel privacy-preserved online training method that can detect malware variants within seconds level. Our extensive experiments on different datasets demonstrate stable training without accuracy loss and 1.39 to 4.43 times speedup compared to the state-of-the-art system.
- We propose MalwareShield, a privacy-preserved federated learning framework, to detect zero-day malware attacks without data privacy concerns. This approach ensures minimal communication costs and expedited training times. The effectiveness of the MalwareShield is compared with state-of-the-art deep learning (DL) methods DeepAuditor [35], ThingNet[46] and DeepShield[39].

1.3 DISSERTATION ORGANIZATION

The remainder of this dissertation is organized as follows. We discuss the necessary background and related work in Chapter 2 and Chapter 3, respectively. Chapter 4 describes the system architecture of the DeepShield model. In Chapter 5, we present our design of the BoTShield model. The zero-day malware attack detection framework MalwareShield is introduced in 6, and Chapter 7 concludes this dissertation.

CHAPTER 2

BACKGROUND

This chapter briefly discusses power auditing, Convolutional Neural Network (CNN), Federated Learning, and Cryptographic tools used in our scheme.

2.1 NON-INTRUSIVE POWER SIDE-CHANNEL MONITORING

using external current sensors has demonstrated effectiveness in extracting malicious signals from IoT device supply power [19], [35]. To instantaneously audit the power consumption, We reuse the current sensors integrated into the IoT device[1]. The current sensor samples the CPU power consumption under various states (i.e., idle, service, and attack). When observing the power pattern during the infection process of the Mirai family, abnormal power spikes occur when Linux commands are executed on the device during the scanning and loading phases. The power data during idle state (e.g., when no service is running on the IoT device), IoT device standard service (e.g., when service is running on the device), and botnet intrusion[2] and infection[3] are illustrated in Figure 1A, Figure 1B and Figure 1C respectively.

To identify distinct power features correlated to malicious infection activities, it is essential to design an effective sliding window approach [16]. The continuous power data stream is partitioned

[1]Without loss of generality, current sensors [74] are integral components of modern heterogeneous multicore architectures for DVFS (dynamic voltage frequency scaling) power management

[2]During the phase of *Scanning* of Mirai, the bot scans open Telnet port of the victim IoT device, launches brute-force attack to obtain login credentials.

[3]Upon discovering the credential, the loader server is commanded by the *C&C* server to gain shell access of the victim device and instruct it to download and execute the malicious binary in the phase of *loading*.

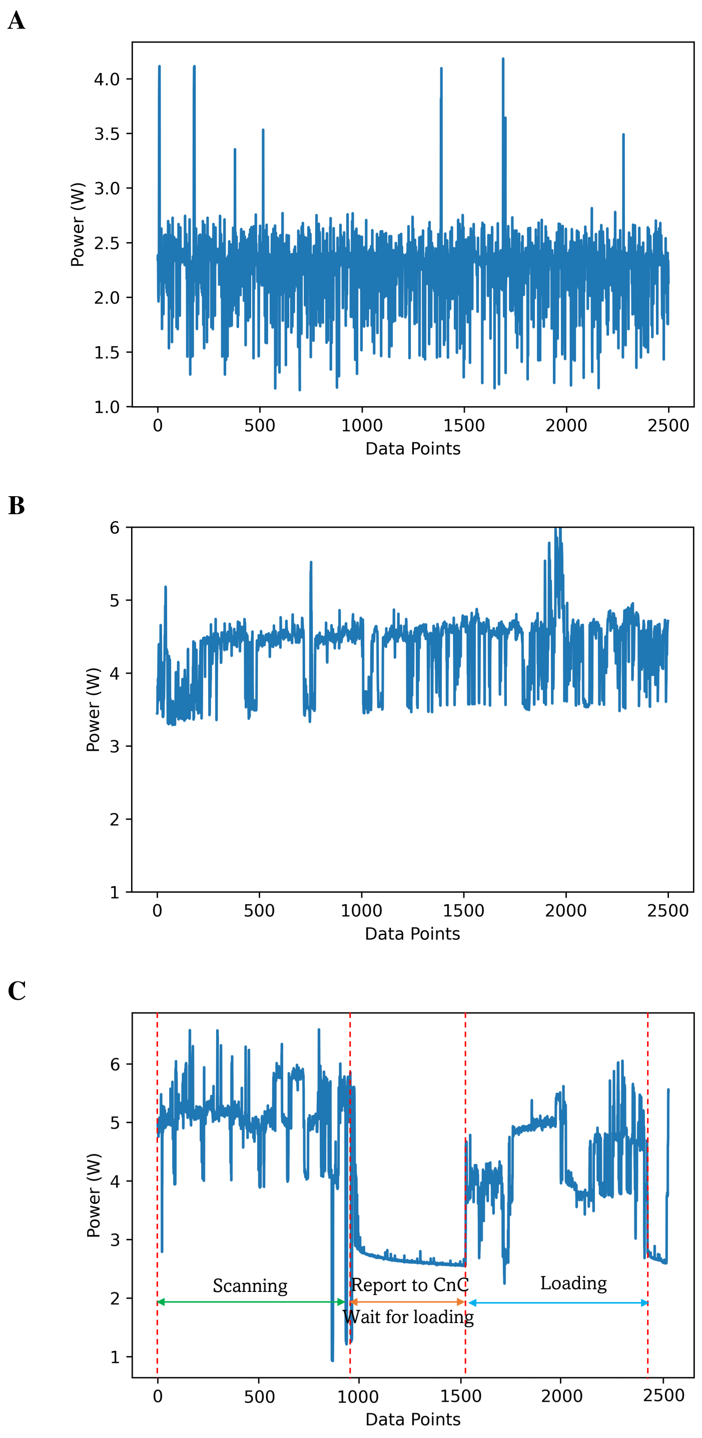


Figure 1. Power data under idle, service and attack. *(A)* When device is idle. *(B)* When device is under service. *(C)* Under botnet intrusion and infection (I would like to thank Zhuoran Li for providing me with this figure).

into separate sequences of botnet infection activities for supervised learning-based detection. A fixed-size overlapping sliding window approach is employed to divide the sequence of events into a set of time-ordered and overlapping sliding windows to achieve better classification accuracy. Aiming to distinguish malicious activities from normal behaviors while identifying identical or similar abnormal events among Mirai variants, the power data stream is segmented according to the Mirai (and its variants) infection events such as scanning, loading, or event transition. Accordingly, the size of the sliding window is set for effective mapping to event labels (e.g., 2.5-second window sizing with $\frac{1}{3}$ overlapping as shown in Figure 1C.

2.2 CONVOLUTIONAL NEURAL NETWORK (CNN)

Convolutional neural network (CNN) is a deep learning network architecture that learns directly from data as depicted in Figure 2. CNNs are effective for classifying effectively time serial signal data, e.g., power patterns [33].

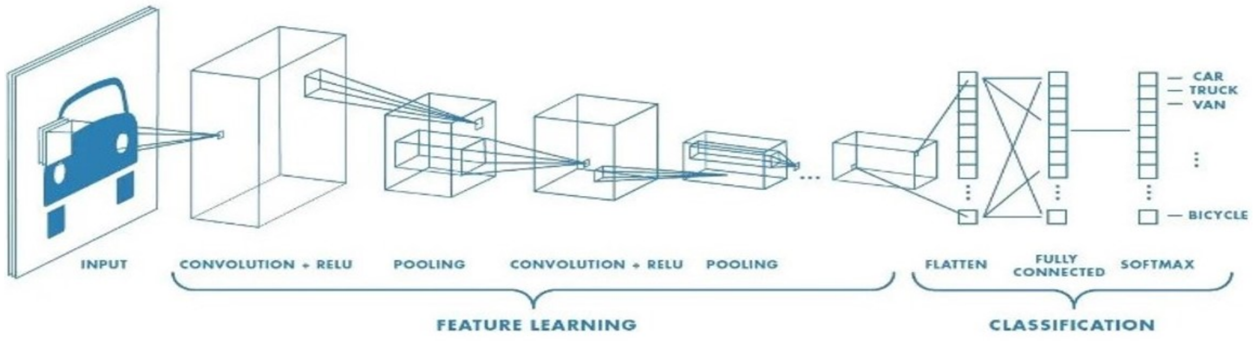


Figure 2. Internal architecture of Convolutional Neural Network (CNN).

The CNN is composed of the following layers:

- Convolutional Layer. In this layer, all neurons share the same weights $\boldsymbol{w}$ and bias $\boldsymbol{b}$, defining a kernel or filter of size n. Each neuron is connected to a specific region of n neurons in the input layer. Denoting n-element $\boldsymbol{x}_i$, each of which is distinctly assigned with one of $\boldsymbol{w}$'s n elements, as the i-th region of input $\boldsymbol{x}$, the i-th element, $\boldsymbol{z}_i$, of output $\boldsymbol{z}$ is computed with the dot products between the filters and local regions of the input that overlaps with the filters, i.e., $\boldsymbol{z}_i = \boldsymbol{w} \cdot \boldsymbol{x}_i + \boldsymbol{b}$.

- ReLU Layer. The activation layer applies elementwise non-linearity to the linear output, e.g., the convolutional layer. The rectified linear unit (ReLU) is used in our CNN model with the activation function $f(x) = \max\{x, 0\}$.

- Max-Pooling Layer. After the ReLU Layer, the pooling layer downsamples the feature maps to minimize computation and manage overfitting. Over the feature maps, max-pooling outputs the maximum value inside a pooling window.

- Fully-Connected Layer. In this layer, every neuron is fully connected to all inputs $\boldsymbol{x}$ from the previous layer. The matrix multiplication is performed between $\boldsymbol{x}$ and the $\boldsymbol{m}$ filters, each of which has the same size as $\boldsymbol{x}$, and the i-th filter and bias is denoted as $\boldsymbol{w}_i$ and $\boldsymbol{b}_i$. Thus the i-th element of the output $\boldsymbol{z}$ is $\boldsymbol{z}_i = \boldsymbol{w}_i \cdot \boldsymbol{x} + \boldsymbol{b}_i$.

2.3 BACKPROPAGATION

An essential artificial neural network (ANN) training strategy is backpropagation, which stands for "backward propagation of errors." This supervised learning method minimizes errors by modifying the neural network weights to reduce the discrepancy between the intended and expected outputs.

The backpropagation works as follows:

Forward Pass (Inference): During the forward pass, the neural network receives layers of input data. After computing a weighted sum of its inputs and applying an activation function, each neuron in a layer sends the output to the layer below. This procedure is repeated until the last output layer generates a prediction.

Error Calculation: When the output is generated, a loss function (e.g., mean squared error for regression or cross-entropy for classification) is calculated to quantify the error, also known as the loss, between the actual target output and the predicted output.

Backward Pass (Backpropagation): The method traverses the network backward in the backward pass, beginning at the output layer. Using the calculus chain rule, the gradient of the loss function for each weight and bias in the network is determined. The gradient shows the relative contributions of each weight and bias to the mistake. These gradients are then utilized to update the network's weights and biases to reduce error in the subsequent training iteration.

Gradient Descent: Gradient descent or its variants (e.g., Adam, stochastic gradient descent) are standard optimization algorithms used with backpropagation to iteratively alter the weights and biases in the direction that minimizes the error. The learning rate determines the gradient descent iteration's step size.

2.4 HOMOMORPHIC ENCRYPTION (HE)

A homomorphic encryption scheme [22] is a public key encryption technique that allows linear operations to be performed directly on the ciphertexts. Specifically, homomorphic encryption consists of a tuple of algorithms $HE = (KeyGenerator, Encr, Decr, Compute)$ with the following syntax:

- $HE.KeyGenerator \to (pk, sk)$. $HE.KeyGenerator$ is a randomized algorithm that outputs a public key pk and a secret key sk.
- $HE.Encr(pk, m) \to c$. The encryption algorithm $HE.Encr$ outputs a ciphertext $c = [m]$ as the encryption of m when the public key is pk and a message m.
- $HE.Decr(sk, c) \to m$. The decryption algorithm $HE.Decr$ outputs the message m contained in $c = [m]$ when the secret key is sk and a ciphertext c,
- $HE.Compute(pk, c1, c2, L) \to c'$. $HE.Compute$ outputs a new ciphertext $c' = [L(m1, m2)]$ encrypting $L(m1, m2)$ after getting the public key pk, two ciphertexts $c1$, $c2$ encrypting messages $m1$ and $m2$, and a linear function L,

As HE operations such as the ones in $HE.Compute$ are expensive, we offload them into offline preprocessing so that the online inference is efficiently computed in plaintext. This contributes to real-time and privacy-preserving detection of IoT Botnet attacks.

2.5 ADDITIVE SECRET SHARING

Given an element x, a 2-of-2 additive secret sharing for x is a pair $([x]_1, [x]_2) = (x - r, r)$ where r is a random number. In this scheme, x can be reconstructed by adding the two shares: $x = ([x]_1 + [x]_2)$. The value x is perfectly hidden when only one of the shares, either $[x]_1$ or $[x]_2$, is known.

2.6 DIFFERENTIAL PRIVACY (DP)

Differential privacy is a concept in data privacy that aims to protect individuals' privacy when their data is used for analysis or research. It provides a mathematical framework and a set of techniques for ensuring that the results of statistical studies do not reveal information about specific individuals in the dataset.

The basic principle of differential privacy is to introduce random noise into the data to preserve the dataset's statistical features while maintaining the privacy of the individual users. This noise is carefully adjusted to strike a compromise between the degree of privacy protection offered and the usability of the data

Differential privacy has become increasingly important in big data, machine learning, and data-driven research, where large datasets are often used to extract valuable insights. By implementing differential privacy mechanisms, organizations and researchers can comply with privacy regulations, gain public trust, and mitigate the risk of re-identification attacks on individuals whose data is included in the analysis.

Overall, differential privacy provides a rigorous framework for achieving privacy-preserving data analysis, ensuring that the benefits of data-driven research and analysis can be realized without compromising the privacy rights of individuals.

2.7 MULTI-PARTY COMPUTATION (MPC)

Several parties can collaboratively compute a function over their inputs while maintaining the privacy of those inputs according to a cryptographic framework called multi-party computation (MPC). Enabling collaborative computation while keeping confidential information hidden from all involved parties is the aim of MPC. The MPC protocol works as follows:

Input Sharing: Every participant enters their data in private. Typically, these inputs are divided into shares or encrypted using cryptographic methods like secret sharing.

Secure Computation: Using cryptographic protocols, the parties collaboratively compute the desired function over their shared inputs. This computation ensures that at no point does any party have access to the complete input data of another party.

Output Reconstruction: After computation, the parties can reconstruct the output of the function without any party learning more than what is revealed by the output itself.

The challenges to building an MPC framework are listed below:

Computational Overhead: MPC protocols can be computationally intensive, requiring careful optimization to scale to large datasets and complex computations.

Communication Complexity: Performing MPC procedures requires effective communication between parties, especially in cases where participants are far out geographically.

Protocol Design: Designing MPC protocols that are secure against various attack scenarios and ensuring interoperability across different platforms and environments.

2.8 FEDERATED LEARNING

Federated learning is a machine learning technique that lets several parties work together to train a common model while maintaining the privacy and decentralization of their data. Federated learning uses locally available data to train local models on each device or server instead of transmitting raw data to a central server for training. Without sharing the raw data, these local models are combined to produce a global model incorporating insights from all the data sources. This decentralized strategy facilitates model refinement through cooperation amongst dispersed data sources while maintaining data confidentiality and privacy.

The benefits of federated learning are coupled with robust privacy protection mechanisms in a privacy-preserved federated learning approach. Model updates from various devices are securely aggregated on a central server without exposing individual contributions. Techniques like secure multi-party computation (SMPC) or homomorphic encryption are deployed to ensure privacy during this aggregation process. Additionally, differential privacy techniques are applied to model updates, introducing noise or randomness to prevent adversaries from deducing sensitive information about individual data points. Data encryption is implemented during transmission and storage, safeguarding it from unauthorized access. Through these privacy-preserving measures, federated learning enables organizations to utilize distributed data sources while upholding privacy and confidentiality standards throughout the collaborative model training process.

2.9 PRIVACY PRESERVING FEDERATED LEARNING (PPFL)

Privacy-preserving Federated Learning (PPFL) is an emerging approach in machine learning that addresses concerns about data privacy and security while leveraging decentralized data sources. It combines the benefits of federated learning with privacy-preserving techniques to enable collaborative model training without the need to centralize sensitive data.

Critical concepts of PPFL are listed below:

Federated Learning (FL): FL is a decentralized approach where multiple parties (often devices or edge nodes) collaboratively train a shared machine-learning model without sharing their raw data with a central server. Instead of sending raw data to a central server, each participant trains a local model using its data and sends only model updates (gradients) to a central aggregator, which then aggregates these updates to update the global model.

Privacy Concerns in FL: Traditional FL can still pose privacy risks because model updates can reveal sensitive information about the local datasets, especially when the aggregator is untrusted or compromised. PPFL aims to mitigate these risks by integrating privacy-preserving techniques into the federated learning process.

Privacy-Preserving Techniques:

Differential Privacy: Adding noise to the gradients before aggregation masks individual contributions and prevents inference of sensitive information.

Secure Aggregation: Using cryptographic techniques such as homomorphic encryption or secure multi-party computation (MPC) to ensure that model updates are aggregated so that the aggregator cannot decipher individual contributions.

Local Model Updates: Limiting the amount of information shared during model updates to only what is necessary for global model improvement while keeping raw data local.

PPFL can be applied in various domains, including but not limited to:

Healthcare: Collaborative training of predictive models using patient data from different hospitals while preserving patient privacy.

IoT: Training models on edge devices without transmitting sensitive data to cloud servers.

Finance: Fraud detection and risk assessment using data from multiple financial institutions without sharing proprietary customer data.

Some challenges in implementing PPFL include:

Computational Overhead: Implementing privacy-preserving techniques can introduce additional computational costs and complexity.

Communication Overhead: Securely aggregating model updates requires efficient communication protocols to minimize latency and bandwidth usage.

Security Concerns: Ensuring the security of cryptographic protocols and protecting against potential attacks on federated learning systems.

CHAPTER 3

RELATED WORK

The security research community has been increasingly focused on the growing menace of IoT malware. While network-based detection remains a prevalent research avenue, there's a shift toward exploring low-overhead side-channel detection solutions. This chapter will discuss the prior work on privacy-preserved neural networks, power monitoring-based malware detection, and the federated learning approach.

3.1 POWER MONITORING BASED MALWARE DETECTION

Several studies have been conducted for IoT security against botnet attacks, mainly network-based approaches [20], [52], [66], [70]. Network-based anomaly detection [54], [57] or signature-based detection [28] has been commonly used for protecting IoT systems. More research is needed to explore the effect of botnet attacks on detailed IoT device behaviors. Malware detection using power consumption monitoring has recently drawn attention for embedded medical devices [13] and mobile devices [40]. A power-signature-based mobile malware-detection approach was proposed in [40] targeting previously unknown energy-depletion threats. WattsUpDoc was designed to monitor the behavior of embedded systems via power side channels for anomaly detection [13]. A code execution tracking-based malware detection scheme was proposed in [49] by observing the power consumption of the microcontrollers. Recent work has detected offline IoT botnet intrusion via power modeling, which relies on an expensive external power monitor [36]. [35] proposes an online intrusion detection system called for IoT devices via power auditing. However, this is an inference model only, so the client's data is not protected during the offline training. Moreover, only a few Mirai variants are used to evaluate the system. We are the first to realize a distributed privacy-preserved online training system for botnet intrusion detection on multiple IoT devices via ubiquitous power auditing. Our work selects the power side-channel signal because it is easy to collect, closely correlated with the system's workload, and can detect the early propagation stage of IoT botnets. Our approach attempts to discover detailed information about the infection activities of IoT malware by fine-grained analysis of its correlation to malicious botnet activities to increase detection accuracy. Additionally, our model preserves the privacy of the client's data and server model even during the training phase, which is the first approach in the intrusion detection domain.

3.2 NEURAL NETWORK INFERENCE AND TRAINING.

In recent years, privacy-preserving machine learning has received considerable research attention. There are four major approaches. The first approach for privacy preservation was introduced by using homomorphic encryption. Perhaps the very first to consider secure neural network prediction was the work of Gilad-Barach *et al.* [21], who used homomorphic encryption techniques to provide a secure prediction. As the non-linear function cannot be computed directly in a privacy-preserved model, they approximated non-linear functions, such as the ReLU activation function, to a quadratic function. But, this simple approximation results in a loss in accuracy; there have been works that approximate ReLU using higher degree polynomials [11], garbled circuit [83] but incur higher cost. CHET [17] is another homomorphic-based inference protocol that replaces all ReLUs with polynomials. Approximations for better efficiency, harming large networks' accuracy. The second approach is a 2PC(secure two-party)-based protocol. SecureML [56] is a 2PC-based protocol that is the first work to provide secure protocols for neural network training and prediction with non-linear activations, using a combination of arithmetic and Yao's garbled circuit techniques. MiniONN [48] uses the SPDZ protocol that further optimizes the protocols of SecureML by reducing the offline cost of matrix multiplications and increasing the online cost. GAZELLE [37] uses an efficient HE-based protocol for linear layers, while garbled circuits compute non-linear activations. However, its reliance on heavy cryptographic operations in the online phase results in a computationally expensive protocol compared to our model. DeepSecure [65], the protocol of Ball *et al.* [5], and XONN [62], all use circuit garbling schemes to implement constant-round secure inference Protocols. The third approach is a 3PC(secure three-party)-based protocol. Chameleon [63], SecureNN [78], ABY [18] and Falcon [43] are 3-PC based protocol. These works have explored how adding a third party can greatly improve efficiency for secure machine learning applications. The fourth approach is TEE(trusted execution enclaves)- based protocols: (a) inference via server-side enclaves, where the client uploads their input to the server's enclave, and (b) inference in client-side enclaves, where the client submits queries to a model stored in the client-side enclave. Slalom [76] and Privado [75] are examples of protocols that rely on server-side enclaves. ML-Capsule [32] describes a system for performing inference via client-side enclaves. TEE-based cryptographic inference protocols offer better efficiency than protocols that rely on cryptography, such as DELPHI, but this improved efficiency comes at the cost of a weaker threat model. In contrast to all the above works, we provide protocols for non-linear activation functions by avoiding garbled circuits, thus dramatically reducing the communication complexity of non-linear layers.

3.3 FEDERATED LEARNING

[7] presented a practical system for secure aggregation in federated learning, ensuring that

individual participant updates remain confidential, even to the aggregator. Similarly, [53] proposed the Federated Averaging (FedAvg) algorithm, which aggregates locally computed updates into a global model without transferring raw data, using cryptographic techniques for the secure summation of encrypted updates. [26] investigated differentially private federated learning, while [44] surveyed various privacy-preserving federated learning methods, summarizing advanced techniques such as homomorphic encryption, secure multiparty computation, and differential privacy. [81] presented a communication-efficient and privacy-protected architecture for federated learning. [80] developed a decentralized trust framework for federated learning, which enhances privacy and security by leveraging blockchain technology and smart contracts to ensure trustworthy and transparent collaboration among participants. [12] explores Privacy-Preserving Federated Learning through Functional Encryption, allowing computations to be performed on encrypted data. This approach ensures that only the final aggregated results are decrypted, keeping individual contributions confidential. Federated learning with non-IID data, as discussed by [86], addresses the challenge of training accurate and robust machine learning models across decentralized datasets with differing distributions.

CHAPTER 4

DEEPSHIELD: LIGHTWEIGHT PRIVACY-PRESERVING INFERENCE FOR REAL-TIME IOT BOTNET DETECTION

4.1 SUMMARY

With the rapid proliferation of IoT devices, botnets have exploited their vulnerabilities. Yet, detecting the initial intrusion on IoT devices before large-scale attacks remains challenging. Recent research has leveraged power side-channel data to spot this intrusion behavior, but real-time, precise models for widespread botnet detection still need to be improved. A key challenge is how to effectively unfold the details of malicious activities executed on the IoT devices to enable fine-grained real-time detection of botnet infection, minimizing the loss of botnet attacks. Deep learning has evolved as a powerful dynamic analysis method of normal and abnormal behavior. However, deploying resource-repletion deep neural networks on resource-constrained IoT devices with intelligent infrastructure and smart cities may be challenging. Though cloud-based deep learning is popular, it raises serious issues of data privacy and response latency. The contribution of our work is two folds: (a) we propose a non-intrusive power side-channel auditing approach leveraging the low-cost current sensors to infer fine-grained analysis of malicious behaviors in IoT botnet infection; (b) we propose DeepShield, a novel lightweight deep neural network online inference model for real-time privacy-preserving feature extraction and classification based on edge computing. The strength of our approach lies in the critical novelty of a hybrid cryptographic protocol that offloads the majority of online computation to the edge and enables secret-sharing

collaborative computation between the smart auditor and edge server. It further takes the most expensive calculation of homomorphic operations offline, lightening online secure interaction. In addition, the non-linear activation is securely outsourced and resolved in an unencrypted form on the client side. We adopt this approach for the non-linear layer because generating and transmitting garbled circuits is time-consuming, particularly for complex data and computation-intensive tasks. We demonstrate that DeepShield can secure high-accuracy, real-time, and scalable botnet infection detection through theoretical analysis and empirical experiments.

4.2 SYSTEM DESIGN OF DEEPSHIELD

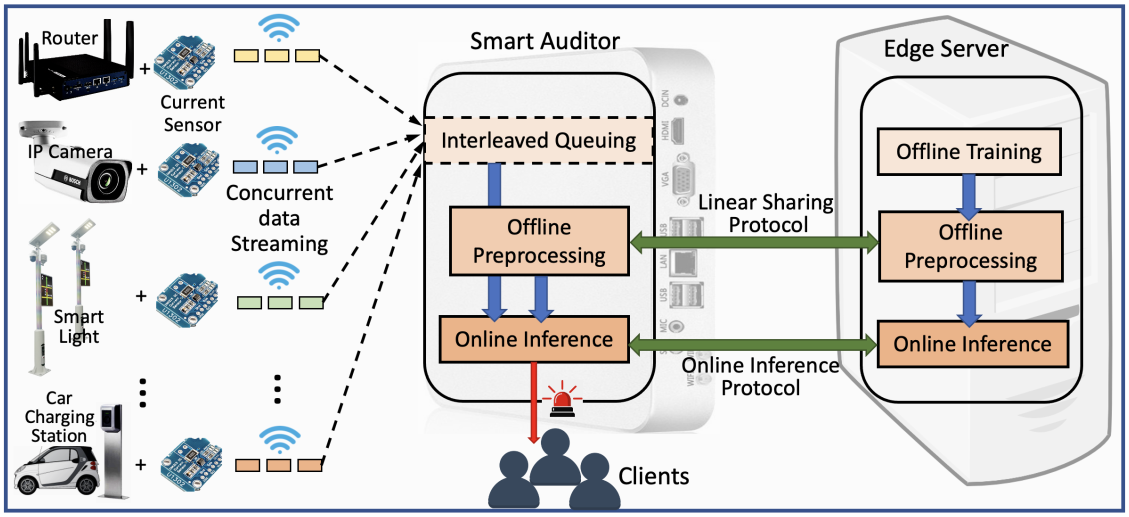


Figure 3. System design of DeepShield (I would like to thank Dr Danella Zhao for providing me with this figure).

Aiming at accurate detection of botnet malware (e.g., Mirai, Mirai variants, LuaBot, Qbot, Ransomware) infection activities by monitoring power fingerprints, we develop an IoT botnet detection system, dubbed *DeepShield*, entailing built-in current sensors and data preprocessing techniques to extract the malicious signal. As shown in Figure 3, the DeepShield system mainly consists of a smart auditor deployed at the client side and an edge server equipped with the MLaaS engine that runs convolutional neural network (CNN) inference for secure IoT botnet detection on the fly.

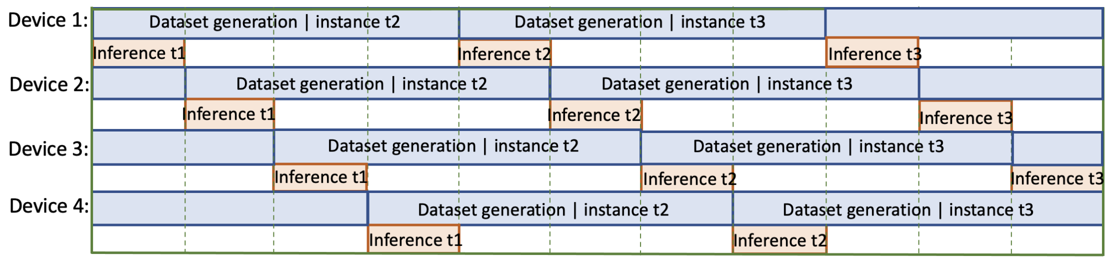


Figure 4. Illustration of interleaved inference per core (I thank Dr Danella Zhao for providing me with this figure).

4.2.1 Smart Auditor and Interleaved Data Streaming

The key to generating datasets is to collect and preprocess suspicious power signals to obtain distinguishable features for activity inference. The *smart auditor* is designed to facilitate efficient data collection, preprocessing, and offline and online computation as involved in the secure interaction protocol described in Section 4.3. During data collection, the power data is sampled via the on-device current sensor and live-streamed to the smart auditor for preprocessing.

System optimization is further performed by exploring scalability. To support real-time monitoring of *N* number of IoT devices in the system (e.g., a smart city surveillance camera system), a

Multi-Stream Interleaved Queuing (MSIQ) scheme is developed to support simultaneous detection of N devices using thread-level parallelism via multicore. We can further speed up detection with hyper-threading and model-level pipelining [79], which are beyond the scope of this work. Without loss of assumption, let the times of power data generation and secure CNN inference be l and m milliseconds, respectively. As the detection procedure is dominated by the dataset generation, i.e., $l \gg m$, real-time system-wide detection can be achieved with a n-core CPU. That is $\frac{nl}{m} \geq N$. A parallel TCP/IP socket interface with multithreading is implemented to support the live streaming power data from N devices. As illustrated in Figure 4, with interleaving and system-level pipelining, we can support concurrent detection on l/m devices with a single core. We will evaluate the system scalability in Section 4.4.

4.2.2 Infection Detection and Offline Training

For high detection accuracy and real-time and low-cost hardware implementation, a $1D$ Convolutional Neural Network (CNN) is developed by learning the correlation between $1D$ device power signals and malware infection activities such as the Linux commands execution during Mirai Brute-force scanning and loading. The edge server offline trains the $1D$ CNN, which consists of a sequence of layers that transform an input layer into an output layer: Convolutional, ReLU, Max-Pooling, and Fully connected Layers. Based on the offline training model, a lightweight CNN inference model named DeepShield is developed to lift the burden of CNN inference off the resource-constrained IoT device while preserving privacy and accuracy. The key idea is the design of a novel online/offline cryptographic protocol, which is discussed in Section 4.3.

4.2.3 Threat Models

Our model can detect botnet infection by non-intrusively analyzing side-channel power data from vulnerable IoT devices. There are two types of attacks that adversaries may launch at the client side: 1) Attacking the power sensor by exploiting its vulnerabilities. 2) Launching complicated post-processing to interfere with detection. Adversaries can perform complicated post-processing jobs to generate different power traces. For example, rebooting the infected device can create longer and more complicated power patterns. To address the former issue, we assume that the IoT devices in a system are equipped with sensor attack detection and mitigation methods [14], [38]. For the later issue, our inference model is trained by collecting power patterns of normal and abnormal activities under various device states, i.e., idle, service, and attack, to detect sophisticated unknown patterns.

Adversaries can also target the client-server-secured interaction protocol. The threat model is similar to what has been discussed in Gazelle [37], SecureML [56], MiniONN [48], and DeepSecure [65] systems. More specifically, DeepShield is designed under a two-party semi-honest setting where the client (denoted as *C*) tries to query the model to learn the model parameters. In contrast, the server (denoted as *S*) wants to understand the privacy-sensitive client data. Our goal is to make *C* oblivious of the model parameters while preventing *S* from obtaining *C*'s private data. We will prove that the proposed model is secure under semi-honest corruption using the ideal/real security analysis [27], [59]. Various attacks have been emerging to comprise neural networks [23], [68], [77]. For example, the *S* can initiate a membership inference attack [68] by comparing *C*'s input with *S*'s pre-trained dataset. *C* can launch the model extraction attack [77] to extract the inference model parameters by exploiting the linear transformation. The model inversion attack [23] utilizes

Scipy's numeric gradient approximation and complete knowledge of softmax probability vectors to find the input that maximizes the classification probability. To counter such an attack, S can return only the predicted label, not the probability vector. Some other attacks try adding small perturbations to the input, leading to misclassification [50], which won't fit our privacy-induced scenario. Protecting the model is usually sufficient through protecting the model parameters, which are the most critical information for a model. Therefore, our protocol only partially hides the whole network architecture; however, we argue that it does hide the vital model parameters that are likely to be proprietary. First, the protocol hides all the weights of the convolution and the fully connected layers. Secondly, the protocol hides the kernel and stride size in the convolution layers and information on which layers are convolutional and fully connected. However, our protocol does reveal the number of layers, the size (the number of hidden nodes) of each layer, and the dimensions of input data. We assume that the sensors and valid users are always available and secure communication channels between entities are available.

4.3 DEEPSHIELD'S SECURE COMPUTATION PROTOCOLS

We develop a series of secure computation protocols based on the additive secret sharing technique to enable secure interaction between the client (smart auditor) and the edge server within our designed DeepShield system (detailed in Section 4.2), aiming to preserve the privacy of both parties while achieving real-time detection. Specifically, a hybrid cryptographic framework is employed to offload computationally demanding cryptographic operations, e.g., homomorphic multiplication and encryption, on the client by an offline preprocessing while utilizing lightweight plaintext operations to streamline an online secure neural network inference. The challenge is enabling the highly computation-efficient and semantically secure two-party computation (2PC) protocols against semi-honest adversaries in the resource-constrained and real-time-demanding IoT botnet detection system to execute online inference without degrading detection accuracy. So, we'd like to elaborate on DeepShield's secure computation protocols for offline preprocessing and online inference in this section.

Protocol 1: Secret Linear Sharing

Client Node	Edge Node
Input: $\boldsymbol{r}^l$	Input: $\boldsymbol{w}^l$, $\boldsymbol{n}^l$
Output: $\boldsymbol{c}^l$	Output: $\boldsymbol{s}^l$ s.t. $\boldsymbol{c}^l + \boldsymbol{s}^l = \boldsymbol{w}^l \cdot \boldsymbol{r}^l - \boldsymbol{n}^l$

Enc($\boldsymbol{r}^l$) ❶	$\overrightarrow{[\boldsymbol{r}^l]}$	Comp($\boldsymbol{w}^l \cdot [\boldsymbol{r}^l] - \boldsymbol{s}^l - \boldsymbol{n}^l$) ❷
		$=[\boldsymbol{w}^l \cdot \boldsymbol{r}^l - \boldsymbol{s}^l - \boldsymbol{n}^l]=[\boldsymbol{c}^l]$
Dec($[\boldsymbol{c}^l]$) ❸	$\overleftarrow{[\boldsymbol{c}^l]}$	

4.3.1 Offline Linear Sharing Protocol

Before executing online inference, the model parameters, e.g., weights and biases, are precomputed by the offline training and stored on the server. During preprocessing, the client and server jointly share a linear output, i.e., the dot product between a noised vector, $\boldsymbol{r}^l \in \mathbb{R}^{m\times 1}$, from the client and the weights, $\boldsymbol{w}^l \in \mathbb{R}^{n\times m}$, from the server, which enables the client to protect its data via that noised vector while producing the correct linear result for the subsequent computation during online inference. Here $\boldsymbol{w}^l$ represents the weight matrix at l-th layer in the Convolutional Neural Network (CNN), e.g., Convolution layer (*Conv*) or Fully Connected (*FC*) layer, and $\boldsymbol{m}$ and $\boldsymbol{n}$ are the input and output dimensions, respectively. The client generates that noised vector $\boldsymbol{r}^l$ and sends it to the server to calculate the share of the linear output $\boldsymbol{w}^l\boldsymbol{r}^l$ as $(\boldsymbol{w}^l\boldsymbol{r}^l - \boldsymbol{n}^l)$, which is then sent back to the client. Here $\boldsymbol{n}^l \in \mathbb{R}^{n\times 1}$ is a randomly generated by server. The secret linear sharing protocol is defined below.

- As illustrated in Protocol 1, to protect the privacy of the random vector $\boldsymbol{r}^l$, client encrypts $\boldsymbol{r}^l$ by Homomorphic Encryption (HE), e.g., packed additive homomorphic encryption (PAHE) technique [37], into $\text{Enc}(\boldsymbol{r}^l) = [\boldsymbol{r}^l]$, which is sent to the server at step ❶.
- Upon receiving $[\boldsymbol{r}^l]$ from client, the server homomorphically computes $\boldsymbol{w}^l \cdot [\boldsymbol{r}^l] = [\boldsymbol{w}^l \cdot \boldsymbol{r}^l]$, e.g., utilizing PAHE based matrix-vector multiplication in [37], where "$\cdot$" denotes the dot product operation[1]. On the other hand, directly sending $[\boldsymbol{w}^l \cdot \boldsymbol{r}^l]$ back to the client could enable the client to deduce the weight parameters from decrypting $[\boldsymbol{w}^l \cdot \boldsymbol{r}^l]$ with high probability [85]. Therefore, as shown in step ❷, the server randomly generates two random vectors, the server's share for $(\boldsymbol{w}^l \cdot \boldsymbol{r}^l - \boldsymbol{n}^l)$ as $\boldsymbol{s}^l \in \mathbb{R}^{n\times 1}$ and $\boldsymbol{n}^l \in \mathbb{R}^{n\times 1}$, and forms the other additive share for client as
$$[\boldsymbol{c}^l] = \boldsymbol{w}^l \cdot [\boldsymbol{r}^l] - \boldsymbol{s}^l - \boldsymbol{n}^l = [\boldsymbol{w}^l \cdot \boldsymbol{r}^l - \boldsymbol{s}^l - \boldsymbol{n}^l].$$
- At step ❸, the server sends $[\boldsymbol{c}^l]$ to client, which decrypts $[\boldsymbol{c}^l]$ into $\text{Dec}([\boldsymbol{c}^l]) = \boldsymbol{c}^l$. It is easy to verify that
$$\boldsymbol{c}^l + \boldsymbol{s}^l = \boldsymbol{w}^l \cdot \boldsymbol{r}^l - \boldsymbol{s}^l - \boldsymbol{n}^l + \boldsymbol{s}^l = \boldsymbol{w}^l \cdot \boldsymbol{r}^l - \boldsymbol{n}^l.$$

  Therefore, at the end of Protocol 1, the client and server each hold a share of $(\boldsymbol{w}^l \cdot \boldsymbol{r}^l - \boldsymbol{n}^l)$ without client knowing server's weight parameters $\boldsymbol{w}^l$ and server knowing client's input $\boldsymbol{r}^l$.

[1]"$\cdot$" can either be homomorphic or plaintext weight-matrix multiplication depending on the involved items are ciphertext or plaintext. Similar logic is applied to $+/-$ in the remaining chapter. The "comp(f)" denotes the computation process defined in f.

It is worth mentioning that Protocol 1 is an offline process independent of real input from the client. Thus, the client can periodically generate new $\boldsymbol{r}^l$, and both client and server can then jointly share the new linear output, $(\boldsymbol{w}^l \cdot \boldsymbol{r}^l - \boldsymbol{n}^l)$, at offline without affecting the performance of online inference as to be discussed next.

4.3.2 Efficient Online Inference Protocol

To facilitate fast online computation without degrading detection accuracy, DeepShield makes the client calculate the nonlinear CNN operations, such as Rectified Linear Unit (ReLu) and Max pooling, using the client's plaintext output of linear computation. The challenge here is how to obliviously perform linear transformation between the two parties such that the client finally obtains the real linear output to subsequently calculate nonlinear operations without knowing the model parameters, e.g., weight matrix and biases. We address this challenge with a secure linear transformation protocol between the client and server, as shown in Protocol 2. Here we denote client's plaintext input as $\boldsymbol{x}^{l-1} \in \mathbb{R}^{m\times 1}$.

Protocol 2: Efficient Online Inference

Client Node		Edge Node
Input: $\boldsymbol{x}^{l-1}$, $\boldsymbol{c}^l$, $\boldsymbol{r}^l$		Input: $\boldsymbol{w}^l$, $\boldsymbol{b}^l$, $\boldsymbol{s}^l$, $\boldsymbol{s}^l$
Output: $f(\boldsymbol{z}^l - \boldsymbol{n}^l)$		s.t. $\boldsymbol{c}^l + \boldsymbol{s}^l = \boldsymbol{w}^l \cdot \boldsymbol{r}^l - \boldsymbol{n}^l$
Comp($\boldsymbol{x}^{l-1} - \boldsymbol{r}^l$) ❶	$\overrightarrow{\tilde{\boldsymbol{x}}^{l-1}}$	Comp($\boldsymbol{w}^l \cdot \tilde{\boldsymbol{x}}^{l-1} + \boldsymbol{b}^l + \boldsymbol{s}^l$) ❷
Comp($f(\tilde{\boldsymbol{z}}^l + \boldsymbol{c}^l)$) ❸ $= f(\boldsymbol{z}^l - \boldsymbol{n}^l)$	$\overleftarrow{\tilde{\boldsymbol{z}}^l}$	

- Recall that the client and server preshare the linear result, $(\boldsymbol{w}^l \cdot \boldsymbol{r}^l - \boldsymbol{n}^l)$, as $\boldsymbol{c}^l$ and $\boldsymbol{s}^l$ by Protocol 1. Therefore, the client first disturbs its input as $\tilde{\boldsymbol{x}}^{l-1} = \boldsymbol{x}^{l-1} - \boldsymbol{r}^l$, which is then sent to server as shown in step ❶. Note that $\boldsymbol{r}^l$ is randomly generated by client, which makes $\tilde{\boldsymbol{x}}^{l-1}$ random to server.

- At step ❷, the server performs a linear transformation, $\tilde{\boldsymbol{z}}^l = \boldsymbol{w}^l \cdot \tilde{\boldsymbol{x}}^{l-1} + \boldsymbol{b}^l + \boldsymbol{s}^l$, and sends $\tilde{\boldsymbol{z}}^l$ back to the client. Note that $\boldsymbol{s}^l$ is randomly generated by server, which makes $\tilde{\boldsymbol{z}}^l$ random to client.

- Upon receiving $\tilde{\boldsymbol{z}}^l$, client recovers the noisy linear result, $\boldsymbol{z}^l - \boldsymbol{n}^l = \boldsymbol{w}^l \cdot \boldsymbol{x}^{l-1} + \boldsymbol{b}^l - \boldsymbol{n}^l$, by adding $\tilde{\boldsymbol{z}}^l$ with $\boldsymbol{c}^l$, as shown in step ❸. It is easy to verify that

$$\tilde{\boldsymbol{z}}^l + \boldsymbol{c}^l = \boldsymbol{w}^l \cdot (\boldsymbol{x}^{l-1} - \boldsymbol{r}^l) + \boldsymbol{b}^l + \boldsymbol{s}^l + \boldsymbol{c}^l = \boldsymbol{w}^l \cdot \boldsymbol{x}^{l-1} + \boldsymbol{b}^l - \boldsymbol{n}^l,$$

  which is a noisy linear output of l-th layer. The nonlinear function f, e.g., ReLu, is then applied to the linear output in plaintext to finally get the nonlinear output, $f(\boldsymbol{z}^l - \boldsymbol{n}^l)$, for l-th layer.

The process in Protocol 2 is repeated until the last layer where client gets the linear result $(\boldsymbol{z}^{l_{last}} - \boldsymbol{n}^{l_{last}})$, and obtains the classification result by choosing the label with maximum in $(\boldsymbol{z}^{l_{last}} - \boldsymbol{n}^{l_{last}})$. In this way, the client and server jointly calculate linear and nonlinear results within one round for each layer in plaintext, enabling efficient and secure detection performance for our DeepShield system.

It's worth mentioning that we add the noise into the CNN network such that the linear result is disturbed by $\boldsymbol{n}^l$. With a negligible drop in detection accuracy (see more details in Section 4.4), the client recovers only the noisy linear result without revealing the actual linear result $\boldsymbol{z}^l$, which effectively prevents the client from deducing the model parameters by accumulating $\boldsymbol{z}^l$ [77]. We have provided a proposition with proof in Section 4.3.3 that DeepShield protects both clients' and servers' sensitive data.

4.3.3 Security Analysis

Proposition 1. The accumulated linear results $\{z_i = w_i a_{i-1} + b_i\}_d$ reveal nothing but the linear combination of weights and bias from which the matrices w_i and b_i cannot be reconstructed by the client.

proof. Let m be the number of neurons, and the function group obtained by the client after one forward propagation is $z^i_{m\times 1} = w^i_{m\times m} a^{i-1}_{m\times 1} + b^i_{m\times 1}$. The function group does not reveal the actual values of the matrices w_i and b_i but the subspaces that are linearly combined by infinitely many possible matrices solutions. Therefore, the client cannot successfully reconstruct model parameters w_i and b_i.

Theorem 1. According to [1] the accumulated noisy linear results $\{\bar{z}_i = w_i a_{i-1} + b_i - n_i\}_d$ reveal nothing if $n_i = N(0, \frac{\sigma^2}{B^2})$, where N is the Gaussian distribution with mean 0 and standard deviation $\frac{\sigma}{B}$, σ is the noise level and B is the mini-batch size.

We also use the ideal/real-world paradigm [59] to prove the security of DeepShield. Specifically, we introduce the following definitions.

Definition 1. A protocol Π securely implements the ideal functionality f^{Ideal} in the semi-honest adversary setting with static corruption if it ensures the following guarantees:

- ***Corrupted server.*** *We require that a corrupted and semi-honest the server does not learn any information about the values in the client's private input x. Formally, there should exist a Probabilistic Polynomial Time (PPT) simulator* $sim_{\mathscr{S}}$ *such that* $view^{\Pi}_{\mathscr{S}} \overset{c}{\approx} sim_{\mathscr{S}}(\boldsymbol{M}, out)$*, where*

$view_{\mathscr{S}}^{\Pi}$ denotes the view of the server in the real protocol execution (including the server's input, randomness, and the transcript of the protocol). $sim_{\mathscr{S}}(\boldsymbol{M}, out)$ is the simulation based on $\mathscr{S}$'s input, i.e., the model parameters $\boldsymbol{M}$ (including weights and bias), and its final output 'out,' e.g., the share of a linear function. The "$\overset{c}{\approx}$" denotes "computationally indistinguishable".

- ***Corrupted client.*** *We require that a corrupted and semi-honest client does not learn any information about the server's model parameters beyond some generic meta-parameters, i.e., the number of input and output channels and the number of layers. Formally, there should exist a PPT simulator $sim_{\mathscr{C}}$ such that $view_{\mathscr{C}}^{\Pi} \overset{c}{\approx} sim_{\mathscr{C}}(\boldsymbol{x}, out)$, where $view_{\mathscr{C}}^{\Pi}$ denotes the view of the client in the real protocol execution (including the client's input, randomness, and the transcript of the protocol). $sim_{\mathscr{C}}(\boldsymbol{x}, out)$ is the simulation based on $\mathscr{C}$'s input, e.g., the input data $\boldsymbol{x}$, and its output 'out', e.g., the result of linear function.*

Based on the above definitions, we show a simulator for different corrupted parties, i.e., the server and the client.

Security against a corrupted client: We define a simulator $sim_{\mathscr{C}}$ for the corrupted client and it conducts as follows:

1. In the offline phase:

- Chooses a uniform random tape for client $\mathscr{C}$, sends a random input $\boldsymbol{r}^l$ to f^{Ideal} and receives the share of $(\boldsymbol{w}^l \cdot \boldsymbol{r}^l - \boldsymbol{n}^l)$ as out.
- Encrypts out with $\mathscr{C}$'s public key, sends the encrypted out to $\mathscr{C}$ and outputs whatever $\mathscr{C}$ outputs.

2. In the online phase:

- Sends the random input $\boldsymbol{r}^l$ and input data $\boldsymbol{x}^{l-1}$ to f^{Ideal} and receives the noised linear result $(\boldsymbol{z}^l - \boldsymbol{n}^l)$ as out.

- Sends $(\boldsymbol{z}^l - \boldsymbol{n}^l)$ to the client and outputs whatever the client outputs.

In the offline phase, $\mathscr{C}$'s view in the real world is the share of $(\boldsymbol{w}^l \cdot \boldsymbol{r}^l - \boldsymbol{n}^l)$, which is identical to that in simulated execution. In the online phase, $\mathscr{C}$'s view in the real world is the noised linear result $(\boldsymbol{z}^l - \boldsymbol{n}^l)$, which is identical to that in simulated execution. As such, the output of $sim_{\mathscr{C}}(\boldsymbol{r}^l, out)$ is computationally indistinguishable to $view_{\mathscr{C}}^{\Pi}$ in offline, and the output of $sim_{\mathscr{C}}(\boldsymbol{x}^{l-1}, out)$ is computationally indistinguishable to $view_{\mathscr{C}}^{\Pi}$ in online. Therefore, the protocol is secure against a corrupted client.

Security against a corrupted server: Similarly, we define a simulator $sim_{\mathscr{S}}$ for the corrupted server $\mathscr{S}$ and it conducts as follows:

1. In the offline phase:

- Sends the model parameter $\boldsymbol{M}$ to f^{Ideal} and gets the None as out.

- Randomly picks a public key, encrypts all-zero input as a ciphertext, and sends it to the server.

- Outputs whatever the server outputs.

2. In the online phase:

- Sends the model parameter $\boldsymbol{M}$ to f^{Ideal} and gets the None as out.

- Randomly picks a vector $\tilde{\boldsymbol{x}}^l$ and sends it to the server.

- Outputs whatever the server outputs.

In the offline phase, the $\mathscr{S}$'s view in real world is client-encrypted ciphertext for input $\boldsymbol{r}^l$ while the one in simulated execution is a ciphertext of all-zero input. In online phase, the $\mathscr{S}$'s view in real world is $(\boldsymbol{x}^l - \boldsymbol{r}^l)$ which is indistinguishable to $\tilde{\boldsymbol{x}}^l$ in simulated execution. As the HE algorithm is secure, the ciphertext in the real world is computationally indistinguishable from that in the simulated execution. Therefore, the protocol is secure against a corrupted server.

4.4 SYSTEM EVALUATION

In this section, we conduct experiments by implementing a test bed prototype of the DeepShield system as shown in Figure 5 to evaluate the performance of DeepShield.

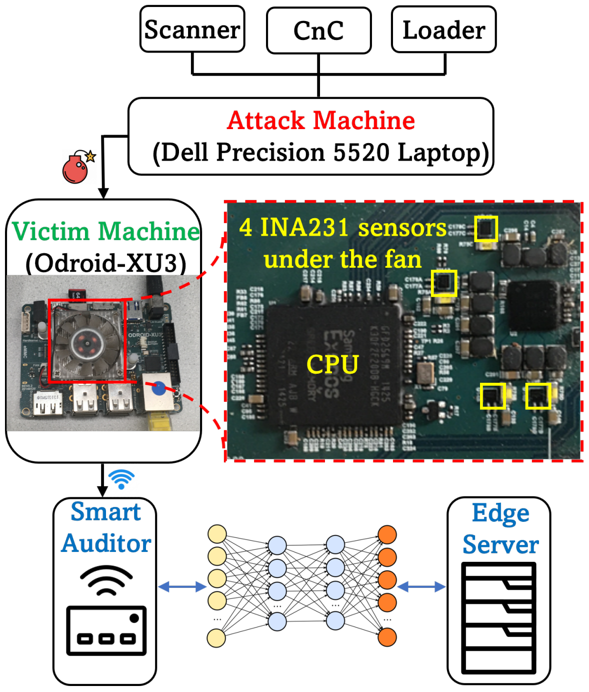


Figure 5. A test bed prototype of DeepShield (I would like to thank Zhuoran Li for providing me with this figure).

The test bed consists of a victim IoT device built from Hardkernel Odroid-XU3, two workstations as a client (smart auditor), an edge server, and an attack machine built on a Dell Precision 5520 Laptop. The client workstation is equipped with a dual-core Intel Core i5-5100 CPU @1.60 GHZ and 6GB RAM, and the server is equipped with a 4-core Intel Core i7-9850H CPU @2.60 GHZ and 16GB RAM. The client and server communicate via a local area network (LAN) with a bandwidth of 1 *Gbps*. Odroid-XU3 has four integrated power consumption monitoring sensors (e.g., TI INA231 current sensor). The CPU power consumption data will be acquired at the highest sampling rate of $1kHz$ under three different device statuses: idle, service, and attack. To launch the botnet attack, the Mirai family botnet scanning, loading, and CnC modules are compiled and installed on the attack machine. We build a test bed prototype of DeepShield to assess its performance.

4.4.1 Data Collection for Malware Family

To validate DeepShield's robustness, power datasets are collected in three different states: Idle (i.e., the IoT device is not running any app or service), IoT service (i.e., the device is running an app), and Botnet attack (i.e., the device is under botnet attack). The model takes the input of power instances and classifies them into one of the three classes.

As given in Table 1, about 12000 instances are acquired for the idle class, 12000 cases for the IoT service class, and 15000 instances for the botnet attack class, respectively, to evaluate the model. We split the dataset into 80:20 rule, so 80% of the data is used for training and 20% for testing. The experiment tested the following malware, namely Mirai, Mirai variants, LuaBot, Remaiten, IRCBot, Qbot, and Ransomware (MedusaLocker).

Table 1. Dataset for our model

Class	Description	Samples
Idle	No service	12000
IoT service	Running an app	12000
Botnet	Mirai	1500
	Mirai Varinats	10500
	Qbot	500
	IRCBot	1500
	Ransomware	1500
	LuaBot	1500

4.4.2 Offline Training and Online Detection Accuracy

The CNN model is trained offline with Tensorflow 2.2, given the training datasets in Table 1. It has four hidden layers, namely convolution, ReLU, max pooling, and the fully connected layer. The input layer has a dimension of 1×2550, and the output is a 3-element vector containing the classification probabilities of the three classes (i.e., idle, service, and botnet attack). Hyper-parameter selection and tuning are then conducted to get the best model. Specifically, the model is fine-tuned by evaluating its performance under different kernel sizes (e.g., 1×32, 1×64, 1×128) and strides (e.g., $1 \times 8, 1 \times 16, 1 \times 32$ of the convolution layer. The number of kernels/filters in the convolution layer is 10. For the max-pooling layer, the pool size and strides are both 1×4. It is essential to highlight that our method does not employ any approximations for the nonlinear computation required in state-of-the-art works. Consequently, the accuracy of offline training and online detection remains consistent.

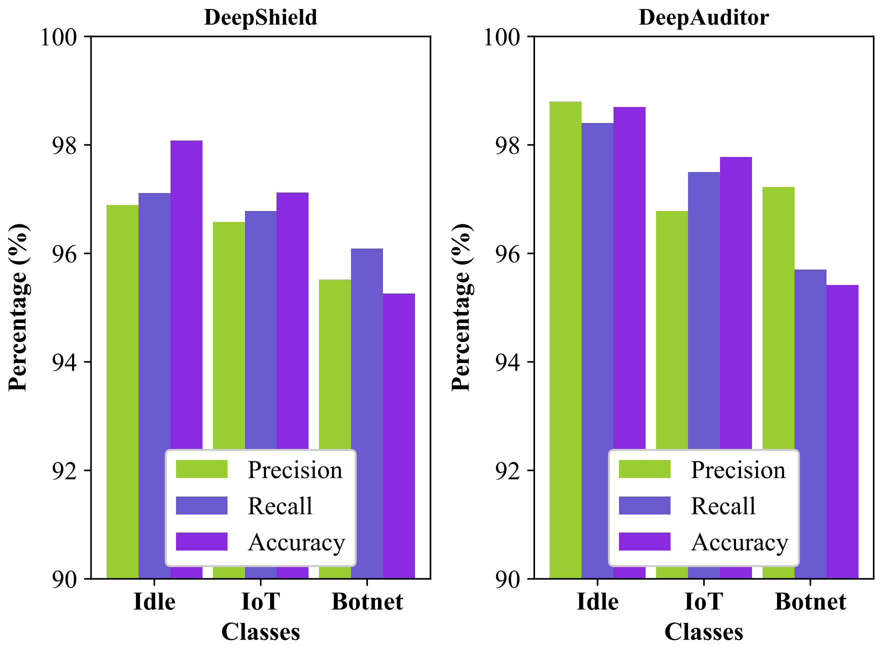


Figure 6. Comparison of online detection accuracy.

In our next experiment, we have compared our model with DeepAuditor [35]. The accuracy, precision, and recall results are summarized in Figure 6 compared with DeepAuditor. As we can see, DeepShield can achieve an overall detection performance of 96.82%, 96.33%, and 96.66% in accuracy, precision, and recall, respectively. We also calibrate the noise level and try to find out the maximum accuracy in DeepShield. In our experiment, DeepShield achieves a maximum accuracy of 88%, 93%, and 96% when different noise levels σ = 8, 4, and 2 are introduced into the model, respectively. When $N(0, \frac{\sigma^2}{B^2}) \approx 0$, then the noise $n_i \approx 0$, resulting in an accuracy of 98.5%, which is comparable to the accuracy of DeepAuditor.

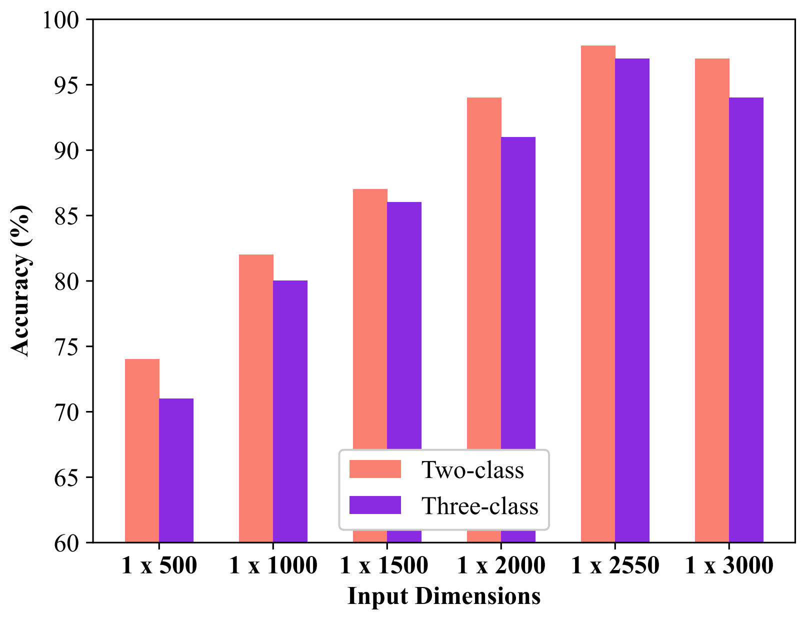


Figure 7. Impact of input dimensions on real-time detection accuracy.

Next, experiments are carried out to fine-tune our model's input dimensions, which are essential for real-time detection. Figure 7 shows that our model cannot converge to a global optimal point when input dimensions are 1×500. This is because this small amount of data does not contain much meaningful information to extract distinguishable power features. When the dimensions increase, the model accuracy improves accordingly, resulting in the best performance at input dimensions of 1×2550.

4.4.3 Microbenchmarks

The runtime performance of linear and nonlinear operations of DeepShield is evaluated and compared to the state-of-the-art secure inference models such as GAZELLE[37], DeepAuditor [35], SecureNN [78] and Muse [42].

Table 2. Comparison of the runtime (ms) and communication cost (MB) of convolution layer

Input	**Kernel**	**Stride**	**Method**	**Time (ms)**		**Communication (mb)**	
				Offline	***Online***	***Offline***	***Online***
1×2550	1×32	8	GAZELLE	—	2033	—	4.044
			DeepAuditor	—	400	—	15
			SecureNN	—	120	0	9.1
			Muse	2600	55	15	1.3
			DeepShield	760	7	8.65	0.045
	1×64	16	GAZELLE	—	3956	—	8.033
			DeepAuditor	—	500	—	18
			SecureNN	—	140	0	11.1
			Muse	2900	72	18	2.3
			DeepShield	784	7.2	8.65	0.032

Linear Computation

As convolution operations are the most computationally expensive, we evaluate the runtime cost of the convolution operation of DeepShield with varying kernel sizes and compare it with other privacy-preserving CNN models. The offline runtime of convolution operations for all three models is defined as the duration when the client encrypts the input data, sends it to the server, and then receives and decrypts the convolution share. The online runtime of DeepShield's convolution operation is the time required for the client to send the plaintext input data to the server and receive the convolution results. The online plaintext computation for linear operation gives DeepShield a substantial performance upgrade compared to other crypto-based approaches. As shown in Table 2, DeepShield demonstrates up to 1461$\times$, 77$\times$, 24$\times$ and 13$\times$ speedup when compared with GAZELLE, DeepAuditor, SecureNN, and Muse under the kernel size of 1×128 and stride of 32.

Meanwhile, the stride-based convolution operation in GAZELLE involves several non-stride computations, which results in a relatively large runtime cost. Furthermore, DeepShield involves an offline process to preshare a random input and calculate HE operations. We adopt this approach to detect malware in real-time. The offline method is data-independent, so it doesn't affect DeepShield's real-time performance. Regarding the communication cost, DeepShield has less transmission load in the online phase as it only involves plaintext data. As such, DeepShield reduces online communication costs over 641$\times$, 600$\times$, 568$\times$ and 128$\times$ compared with GAZELLE, DeepAuditor, SecureNN, and Muse with the kernel size of 1×128 and stride of 32.

Next, we evaluate in Table 3 DeepShield's runtime cost of dot product operation compared with GAZELLE, DeepAuditor, SecureNN, and Muse. Similarly, we define the offline runtime of dot product operation for all four models as the duration when the client encrypts the input data, sends

Table 3. Comparison of runtime and communication cost of fully connected layer

Input	**Method**	**Time (ms)**		**Comm.(mb)**	
		Offline	***Online***	***Offline***	***Online***
1×790	GAZELLE	—	143	—	0.506
	DeepAuditor	—	120	—	0.506
	SecureNN	—	1.2	—	.012
	Muse	220	0.41	4.5	0.025
	DeepShield	95	0.08	1.38	0.006
1×390	GAZELLE	—	141	—	0.503
	DeepAuditor	—	105	—	0.398
	SecureNN	—	0.9	—	.008
	Muse	180	0.29	3.2	0.015
	DeepShield	83	0.06	1.38	0.003

it to the server, receives and decrypts the dot product share. The online runtime of DeepShield's dot product operation is the time required for the client to send the plaintext input data to the server and receive the dot product results. DeepShield shows its significant advantage of online runtime with up to 2700×, 1960×, 14×, and 4.4× speedup compared with GAZELLE, DeepAuditor, SecureNN, and Muse when the input dimensions of the fully connected layer are 1×190. The fundamental advantage lies in DeepShield's plaintext computation in online reference.

Table 4. Comparison of runtime of non-linear layers

Input	**Method**	**Time (microseconds)**	
		ReLU	***Max-pooling***
315×10	GAZELLE	1020000	1238000
	DeepAuditor	113000	125000
	SecureNN	133000	145000
	Muse	99800	102000
	DeepShield	192	687

Non-Linear Computation

We finally examine DeepShield's performance of nonlinear computation, i.e., ReLU and Max pooling, and compare the results with GAZELLE, DeepAuditor, SecureNN, and Muse as given in Table 4. GAZELLE uses the Garbled Circuits (GC) based protocol to obtain the nonlinear result, where computation cost is proportional to the input dimension with multiple communication rounds between client and server. GAZELLE's cost for nonlinear computation is much higher than that of DeepShield, as the GC-based protocol is more expensive than DeepShield's plaintext computation.

Scalability

Last but not least, we evaluate the scalability of DeepShield. The power dataset generation time (around $\sim 833ms$) dominates the detection latency, whereas online inference only takes about $8ms$.

By applying interleaving and pipelining, $\sim 104+$ IoT devices can be detected simultaneously in real time with a single-core hardware architecture. By applying multi-threading assisted by multi-core, the system can support concurrent detection of $104n+$ devices with a n-core configuration while the detection time remains the same.

Our scheme outperforms previous work by several orders of magnitude regarding the runtime. This is mainly because we don't rely on any heavy cryptographic primitives. In addition, generating and transmitting garbled circuits are time-consuming; thus, we refrain from using garbled circuits in the non-linear layers. It is evident from the above experiments that the state-of-the-art listed in Section 4.4 fails to detect IoT malware attacks in real time, whereas DeepShield can detect IoT botnets with high accuracy because of its advantageous design.

CHAPTER 5

BOTSHIELD: PRIVACY-PRESERVING NEURAL NETWORK TRAINING AND MALWARE DETECTION AT IOT-EDGE

5.1 SUMMARY

The proliferation of Internet of Things (IoT) devices, with a projection of 75 billion by 2025, has made them the prime targets for cybercriminals [51]. The characteristics of IoT devices, operating with limited resources and lacking proper defense mechanisms, make them vulnerable to exploitation by attackers. Malware detection for IoT devices by using side-channel data analysis and deep learning techniques, as indicated by studies [6], [19], [36], [46], have shown promise in identifying malicious activities. However, owing to the rapid evolution of malware, traditional offline-trained on-device malware detection methods need to catch up with the swift evolution of the current IoT environment. Recent reports in [4] have revealed that 560,000 new malware are detected daily, equating to more than six new malware variants generated every second. Online training with real-time collected malware data can provide considerable advantages in detecting IoT malware and protecting IoT devices effectively. This work is primarily motivated by the need to develop an online training malware detection strategy capable of keeping up with the fast pace of malware evolution. Previous studies have predominantly focused on either offline training [58] or on-device online training with limited functionality [47]. However, these approaches must be better suited for large-scale infrastructures, such as major global businesses and governments, which often encompass hundreds or thousands of IoT devices within their networks. Hence, our secondary motivation is to devise a large-scale malware detection method capable of simultaneously providing malware detection services for all connected IoT devices. Therefore, we propose BoTShield, a distributed privacy-preserved online training model for inferring malicious activities by analyzing power side-channel signals from IoT devices. We use a hybrid cryptographic approach with secret sharing to protect the privacy of the client's data and the server model parameters during forward propagation. To maintain privacy during backpropagation, we safeguard gradients from the server by adding a mask to the gradients before the client transmits them to the server. In this way, the server is oblivious to the actual weights and cannot figure out the activations. Through theoretical analysis and empirical experiments, we demonstrate that BoTShield can detect infection activities of different IoT malware with high accuracy in real time. Our extensive experiments on various datasets demonstrate stable training without accuracy loss and 1.39 to 4.43 times speedup compared to the state-of-the-art system. Moreover, these techniques allow us to achieve an 8 to 500 times improvement in BoTShield's inference prediction latency compared to the state-of-the-art prior work.

5.2 SYSTEM DESIGN OF BOTSHIELD

BoTShield, a dedicated IoT malware detection system, achieves accurate and practical detection of open-world malware infections by auditing power side-channel fingerprints. BoTShield comprises multiple smart gateways for data streaming, collection, and preprocessing, along with a novel secure online training module for the client and server, as depicted in Figure 8.

5.2.1 Smart Gateway and Multi-Thread Data Streaming, Collection and Preprocessing

The smart gateway/client is connected to each IoT device in the user site to collect power con-

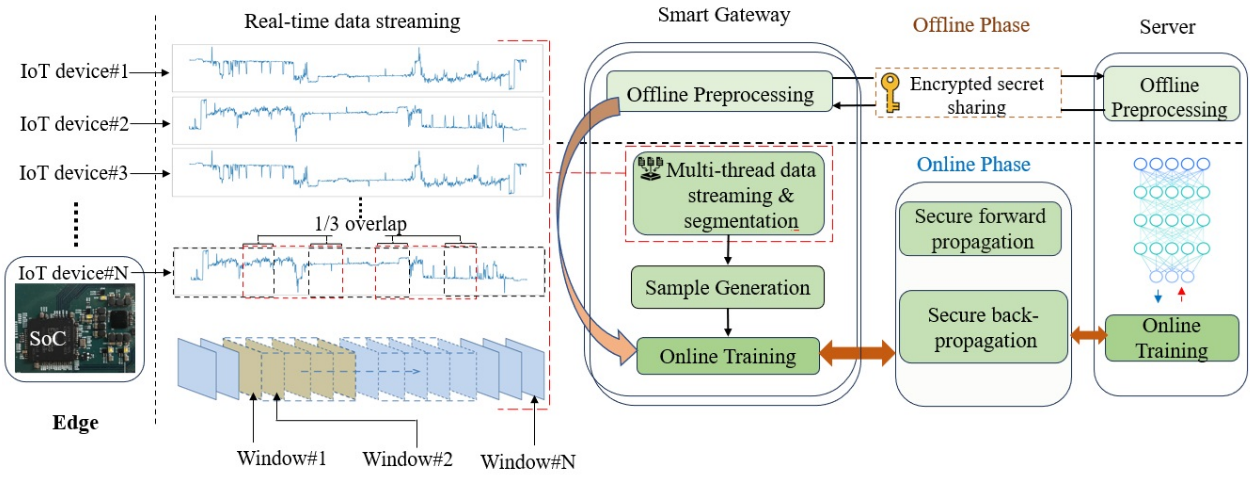


Figure 8. System design of BoTShield (I would like to thank Zhuoran Li for providing me with some parts of this figure).

sumption traces. The core of dataset generation lies in collecting and preprocessing power side-channel signals to derive representative features essential for malware infection detection. We use multiple smart gateways to support data collection from various IoT devices. The smart gateway is designed with a focus on two purposes: firstly, it is designed to efficiently and practically collect data through multi-thread data streaming; secondly, it incorporates a secure offline and online training capability, as elaborated in Section 5.3. During the data collection, the power side-channel data is sampled via the on-device current sensor and live-streamed to the smart gateway for preprocessing. Subsequently, the collected power side-channel data is segmented with appropriate sliding window size and overlap ratio and then pushed into a local queue, awaiting retrieval as an instance for the training module. We use sliding window protocol because overlapping input instances of the Convolutional Neural Network (CNN) classifier can help to achieve better detection accuracy. However, if the smart gateway sends those overlapping windows to the server sides, this will lead to networking redundancy. Hence, 0.5 seconds of segmented data is sent to the smart gateway to generate input instances. After receiving three consecutive packets, the smart gateway assembles the last three packets and sends them to the server for online training. To facilitate real-time monitoring of a system comprising multiple IoT devices, such as an intelligent city surveillance camera system, we've developed a multi-thread data stream strategy that supports the simultaneous detection of various devices through thread-level parallelism across multi-core.

5.2.2 BoTShield's Secure Online Training Overview

The bottleneck of secure online training is the highly costly computation time and accuracy loss; therefore, we adopt a multiple-client, one-server paradigm for our system to reduce latency. We discuss in Section 5.4 that reducing the latency to 0.5 seconds is only possible with multiple clients and single server models. The most computationally intensive operation in our model is cryptographic operations. So, we would like to offload these operations in the offline phase. The smart gateway doesn't receive the power consumption data during the offline phase. That means we conduct the offline phase before the IoT devices send the data. In this phase, we introduce a novel resource-wise algorithm, where randomly generated noise is utilized as a critical component to ensure privacy during forward propagation between the client and server. To ensure privacy, the noise is encrypted via homomorphic encryption and shared between both parties during the offline phase. During the online training process of forward propagation, our model uses these precomputed cryptographic operations; thus, heavy cryptographic computations are bypassed during the online phase to enable real-time detection during the inference phase. To avoid accuracy loss and reduce computational cost, we didn't use approximation functions such as garbled circuits to imitate nonlinear activation functions. Hence, the nonlinear activation is securely outsourced and resolved in an unencrypted form on the client side. To maintain privacy during backpropagation, we safeguard gradients from the server. Exposing private data, such as gradients, to the server poses a risk of privacy breaches. An innovative approach is designed in which the client employs random masking on the gradients before transmitting them to the server. In this way, the server is oblivious to the actual weights and cannot figure out the activations. The client removes the random mask before nonlinear activation to recover the original values post-activation. More details are discussed in Section 5.3.

5.2.3 Threat Models

The threat model is the same as Subsection 4.2.3.

5.3 DESIGN OF BOTSHIELD TRAINING

This section introduces the BoTShield training algorithm for multi-party semi-honest settings. We propose using resource-intensive homomorphic encryption for offline secret sharing and efficient algorithms for online training. Subsections cover offline secret sharing, secure forward propagation, and backpropagation, featuring a novel approach using random noise and gradient masking for privacy preservation.

5.3.1 Offline Secret Sharing

The BoTShield's forward propagation framework employs offline secret sharing and an online phase, as depicted in Figure 9 and Figure 10, respectively. The client input c_i^b serves as a random masking vector, while the server inputs consist of initial weight w_i^I, initial bias b_i^I, and random masking vector s_i^b. Here, i denotes a linear layer, b denotes the batch size, and n is a

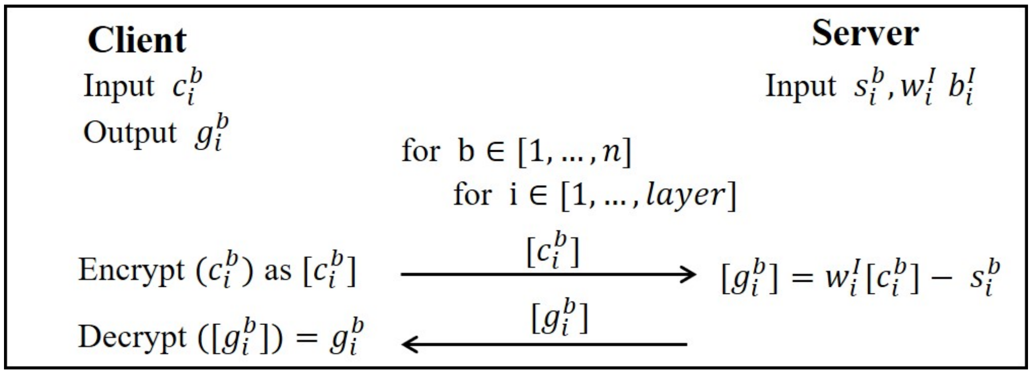
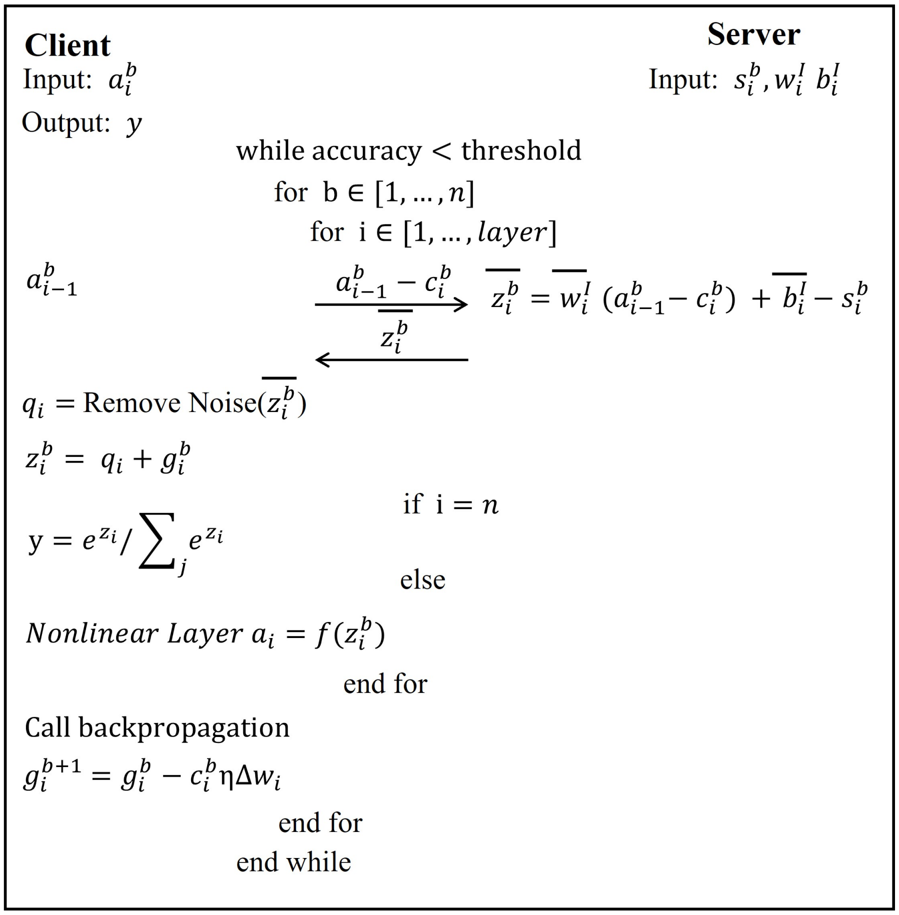

Client
Input c_i^b
Output g_i^b
Server
Input $s_i^b, w_i^I\ b_i^I$
for b ∈ [1, …, n]
for i ∈ [1, …, $layer$]
Encrypt (c_i^b) as [c_i^b]
[c_i^b]
[g_i^b] = w_i^I[c_i^b] − s_i^b
Decrypt ([g_i^b]) = g_i^b
[g_i^b]

Figure 9. BoTShield's offline phase.

server-imposed bound. During the offline phase, the client generates public and secret keys (*pk*, *sk*), encrypts c_i^b with *pk*, and sends it to the server. The server homomorphically computes the weighted sum $w_i^I[c_i^b] - s_i^b$ and sends the resulting ciphertext back to the client. The client decrypts this encrypted weighted sum, yielding $g_i^b = w_i^I c_i^b - s_i^b$ for each linear layer. The server holding s_i^b for a specific layer enables the client and server to possess an additive secret sharing of $w_i^I c_i^b$ for that particular layer. This approach directly executes linear operations on secret-shared data during online processing, bypassing resource-intensive cryptographic operations. This optimization significantly enhances BoTShield's computational efficiency, facilitating real-time detection during inference when power data is accessible.

5.3.2 Secure Forward Propagation

During the online phase, in the input layer, the client sends a real input vector (power data) a_{i-1}^b to the server by introducing noise c_i^b. Subsequently, the server executes the linear layer operation, a linear transformation denoted as $\bar{z}_i^b \leftarrow [\bar{w}_i^I(a_{i-1}^b - c_i^b) + b_i^I - s_i^b]$, and forwards $\bar{z}_i^b$ to the client. Here, w_i^I and b_i^I denote the initial weights and biases of the model. Since nonlinear operations are



```
Client                                              Server
Input: $a_i^b$                                      Input: $s_i^b, w_i^I\ b_i^I$
Output: $y$
            while accuracy < threshold
              for b $\in [1, \dots, n]$
                for i $\in [1, \dots, layer]$
$a_{i-1}^b$        $\xrightarrow{a_{i-1}^b - c_i^b}$   $\overline{z_i^b} = \overline{w_i^I}\ (a_{i-1}^b - c_i^b) + \overline{b_i^I} - s_i^b$
                   $\xleftarrow{\overline{z_i^b}}$
$q_i$ = Remove Noise($\overline{z_i^b}$)
$z_i^b = q_i + g_i^b$
                    if i = $n$
$y = e^{z_i} / \sum_j e^{z_i}$
                    else
$Nonlinear\ Layer\ a_i = f(z_i^b)$
                end for
Call backpropagation
$g_i^{b+1} = g_i^b - c_i^b \eta \Delta w_i$
              end for
            end while
```

Figure 10. BoTShield's online forward propagation phase.

performed on the client side, it becomes necessary to eliminate noise from $\bar{z}_i^b$. Consequently, the client invokes Figure 13 method to obtain q_i and adds it to $g_i^b = w_i c_i^b - s_i^b$ to eliminate noise from $\bar{z}_i^b$, followed by performing nonlinear operations at the client. This process repeats until it reaches the softmax layer, after which Figure 11 method initiates secure backpropagation. Note that as the client recovers $z_i^b = w_i^I a_{i-1} + b_i^I$, the client can accumulate a_{i-1}^b and z_i^b for every $i \in [layer]$ over several iterations to solve the linear equation $w_i^I a_{i-1}^b + b_i^I = z_i^b$ for w_i^I and b_i^I. In a layer with p neurons, there are $p^2 + p$ unknowns, consisting of p^2 weighted connections and p biases. During a single iteration, a set of p linear equations can be established, potentially enabling the client to extract the model after $p+2$ iterations, which could lead to data leakage [77]. To mitigate this risk, the server introduces a threshold n, randomly selected from the interval $(1, p+2)$, which sets an upper limit on how long a client's data can be continuously used for training. Once n is reached, the next client is selected. If training requires more iterations than n, the server can return to the same client for future training sessions without exceeding the n limit.

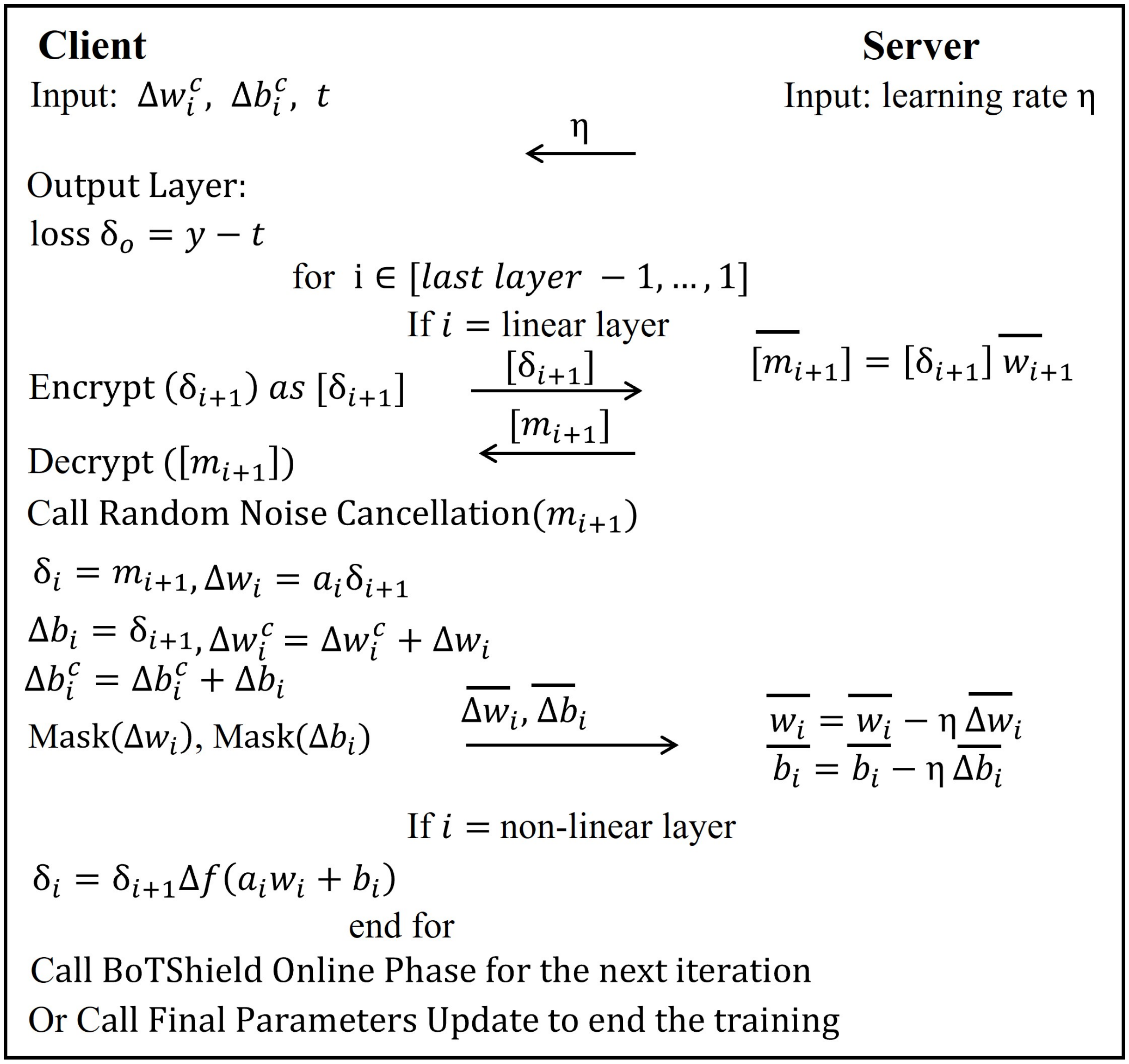
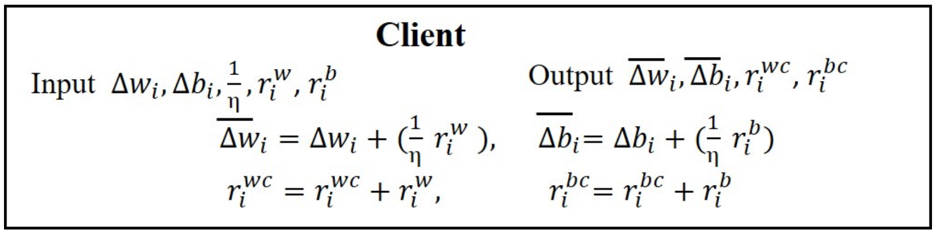

Client
Server
Input: $\Delta w_i^c,\ \Delta b_i^c,\ t$
Input: learning rate η
η
Output Layer:
loss $\delta_o = y - t$
for i $\in [last\ layer\ - 1, \ldots, 1]$
If i = linear layer
Encrypt (δ_{i+1}) as $[\delta_{i+1}]$
$[\delta_{i+1}]$
$\overline{[m_{i+1}]} = [\delta_{i+1}]\,\overline{w_{i+1}}$
$[m_{i+1}]$
Decrypt $([m_{i+1}])$
Call Random Noise Cancellation(m_{i+1})
$\delta_i = m_{i+1}, \Delta w_i = a_i \delta_{i+1}$
$\Delta b_i = \delta_{i+1}, \Delta w_i^c = \Delta w_i^c + \Delta w_i$
$\Delta b_i^c = \Delta b_i^c + \Delta b_i$
Mask(Δw_i), Mask(Δb_i)
$\overline{\Delta w_i}, \overline{\Delta b_i}$
$\overline{w_i} = \overline{w_i} - \eta\,\overline{\Delta w_i}$
$\overline{b_i} = \overline{b_i} - \eta\,\overline{\Delta b_i}$
If i = non-linear layer
$\delta_i = \delta_{i+1} \Delta f(a_i w_i + b_i)$
end for
Call BoTShield Online Phase for the next iteration
Or Call Final Parameters Update to end the training

Figure 11. Secure backpropagation.

5.3.3 Secure Backpropagation

In Figure 11, the backpropagation process commences at the softmax layer, where the error δ_i between the output y and the true label t is calculated. This error propagates through the network to adjust weights w_i and biases b_i across all linear layers using gradients Δw_i and Δb_i. However, to safeguard client data privacy, δ_i is encrypted before transmission to the server for computing $[\bar{m}_{i+1}] = [\delta_{i+1}]\bar{w}_{i+1}$. Subsequently, the server returns $[\bar{m}_{i+1}]$ to the client for decryption and noise removal, facilitating input gradient computation for later use. Weight and bias updates follow through backpropagation equations: $w_i = w_i - \eta \Delta w_i$ and $b_i = b_i - \eta \Delta b_i$, where η denotes the server's learning rate. The gradients of linear layers, Δw_i and Δb_i, are determined as $\Delta w_i = a_{i-1}\delta_i$ and $\Delta b_i = \delta_i$, where a_i signifies the activation from the i-th layer. For non-linear layers, δ_i is updated as $\delta_i = \delta_{i+1}\Delta f(a_i w_i + b_i)$ to use in linear layer gradient updates. During the linear layer computation, weight and bias updates occur on the server side, but the server should not have access to the actual gradients to prevent privacy leaks. To address this, the client invokes the


Client

Input $\Delta w_i, \Delta b_i, \frac{1}{\eta}, r_i^{w}, r_i^{b}$ Output $\overline{\Delta w}_i, \overline{\Delta b}_i, r_i^{wc}, r_i^{bc}$

$$\overline{\Delta w}_i = \Delta w_i + (\frac{1}{\eta} r_i^{w}), \quad \overline{\Delta b}_i = \Delta b_i + (\frac{1}{\eta} r_i^{b})$$

$$r_i^{wc} = r_i^{wc} + r_i^{w}, \quad r_i^{bc} = r_i^{bc} + r_i^{b}$$

Figure 12. Mask the gradient.

Figure 12 method to mask the gradients as $\Delta\bar{w}_i$ and $\Delta\bar{b}_i$ before sending them to the server. The injected randomness sums r_i^{wc} and r_i^{bc} are also computed in Figure 12. To ensure the recovery of original values after activation in non-linear layers and prevent the accumulation of masked errors, the client invokes Figure 13 method in each iteration, extracting m_{i+1} from the masked representation $\bar{m}_{i+1} = \delta_{i+1}\bar{w}_{i+1}$. Additionally, the client tracks cumulative true gradients Δw_i^c and Δb_i^c, along with injected randomness sums r_i^{wc} and r_i^{bc}, crucial for final model parameter update, as shown in Figure 14.

5.3.4 Secure Weight and Bias Gradient Updates

To protect the gradients from the server during backpropagation and prevent the potential derivation of activations and client data, the client adds random vectors to the gradients before transmitting them. This method is inspired by the potential risk of the server reverse-engineering

Client

If forward propagation

Input $\overline{z_i}, a_{i-1}, r_i^{wc}\ r_i^{bc}, r_i$ Output q_i

$q_i = \overline{z_i} + r_i^{wc} a_{i-1} + r_i^{bc} - r_i^{wc}\ r_i$

If backpropagation

Input $\overline{m}_{i+1}, \delta_{i+1}, r_{i+1}^{wc}$ Output m_{i+1}

$m_{i+1} = \overline{m}_{i+1} + \delta_{i+1} r_{i+1}^{wc}$

Figure 13. Random noise cancellation.

activations and client data using equations like $\Delta w_i = a_{i-1}\delta_i$ and $\Delta b_i = \delta_i$, which could compromise client privacy, and therefore lately by the utilization of techniques like the Moore-Penrose inverse [10] in fully connected networks to estimate client data a through operations such as $\tilde{a} = w^T(ww^T)^{-1}(z-b)$, where $z = f^{-1}(a_1)$ represents the inverse of the activation function. With the secure gradient updates method, the client generates uniformly distributed random vectors r_i^w and r_i^b over $\mathbb{R}^n$, where $i \in [layer]$. These random vectors are incorporated into the gradients using Figure 12 method, updating the gradients as $\Delta\bar{w}i = \Delta w_i + (1/\eta r_i^w)$ and $\Delta\bar{b}i = \Delta b_i + (1/\eta r_i^b)$. The

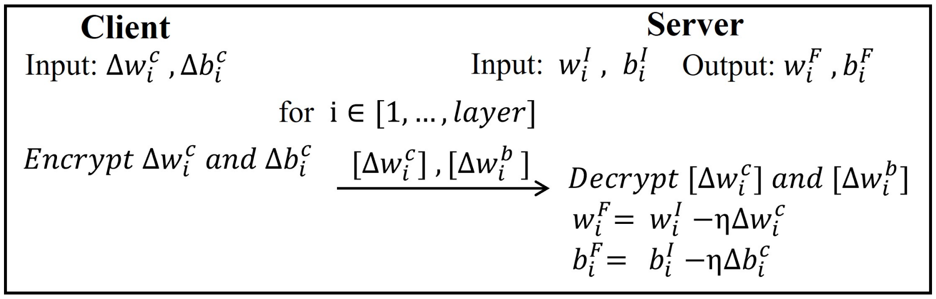

Client
Server
Input: Δw_i^c , Δb_i^c
Input: w_i^I , b_i^I Output: w_i^F , b_i^F
for $i \in [1, \ldots, layer]$
$Encrypt\ \Delta w_i^c\ and\ \Delta b_i^c$
$[\Delta w_i^c]$, $[\Delta w_i^b]$
$Decrypt\ [\Delta w_i^c]\ and\ [\Delta w_i^b]$
$w_i^F = w_i^I - \eta\Delta w_i^c$
$b_i^F = b_i^I - \eta\Delta b_i^c$

Figure 14. Final parameters update.

server updates the model parameters without knowing the true gradients as

$$\begin{aligned} \bar{w}_i &= \bar{w}_i - \eta(\Delta w_i + r_i^w/\eta) = \bar{w}_i - \eta\Delta w_i - r_i^w, \\ \bar{b}_i &= \bar{b}_i - \eta(\Delta b_i + r_i^b/\eta) = \bar{b}_i - \eta\Delta b_i - r_i^b \end{aligned} \tag{1}$$

In this manner, the client ensures that the actual weights and biases remain concealed from the server, preventing it from deducing the activations. Notably, during forward propagation, the client removes accumulating random noises from $\bar{z}$ and supplements $\delta_{i+1} r_{i+1}^w$ to $\bar{m}_{i+1}$ during backpropagation to eliminate noises and restore original values post-activation as shown in Figure13. The server remains unaware of the actual weights throughout the training process, so the accurate weights must be updated with the server upon completion. Since the gradients are masked with random vectors, the server updates the final weights in one step by subtracting the cumulative sum of actual gradients, as depicted in Figure 14.

5.3.5 Security Analysis

In this section, we perform a security analysis of our model, specifically assessing its resilience against well-known equation-solving attacks within the semi-honest model [77].

Proposition 1. The accumulated linear results $\{z_i = w_i a_{i-1} + b_i\}_d$ reveal only the linear combination of weights and biases, making it impossible for the client to reconstruct the matrices w_i and b_i when $b \leq n$.

proof. Let p be the number of neurons, and the function group obtained by the client after one forward propagation is $z^i_{p\times 1} = w^i_{p\times p} a^{i-1}_{p\times 1} + b^i_{p\times 1}$. As long as the client is trained for $b <= n$, the function group does not reveal the actual values of the matrices w_i and b_i, but infinitely many possible matrices solutions linearly combine the subspaces. Therefore, model parameters w_i and b_i cannot be successfully reconstructed by the client with $b \leq n$.

Proposition 2. The server is oblivious to the actual w_i and b_i, so it cannot take input from one party, generate an output, and figure out the activation and client's data.

proof. The ideal and real world paradigm [59] is utilized to prove this proposition. The fundamental concept revolves around a Probabilistic Polynomial Time (PPT) simulator, which can take input from one party, generate an output, and demonstrate that it gains no knowledge other than the outcome. Let the client select r_j, and r_k be selected by a server such that both r_j and $r_k \in \mathbb{R}^n$, here $\mathbb{R}$ is a finite ring. The probability that r_j equals r_k is bounded by $Pr_j = r_k \leq 1 - e^{-2/|\mathbb{R}^n|}$ [67], where $|\mathbb{R}^n|$ denotes the size of a finite field. Given the independence of the random masks, the server can accurately construct matrices of r^w_i and r^b_i with probabilities $Pr = r^w_i \leq (1 - e^{-2/|\mathbb{R}^n|})^{m^2}$

and $Pr = r_i^b \leq (1 - e^{-2/|\mathbb{R}^n|})^m$. Given that $|\mathbb{R}^n|$ is a large value, the probability of the server successfully deriving the gradients approaches zero. Consequently, the server remains oblivious to the actual weights and biases, making it incapable of deducing the activations and the client's data.

Proposition 3. The final parameter update groups $\{w_i^F = w_i^I - \eta \Delta w_i^c, b_i^F = b_i^I - \eta \Delta b_i^c\}_i$ reveal nothing but the subspaces of gradients from which the matrices Δw_i and Δb_i in the previous iteration cannot be reconstructed.

proof. In the final parameter update, the client transmits the summation of the correct gradients Δw_i^c and Δb_i^c to the server. Given that we have set n as the upper limit for client training iterations, the server obtains $n-1$ sets of randomized gradients $\Delta \bar{w} i$ and $\Delta \bar{b} i$. Regarding the weight and bias matrices, a total of n linear equations exist for each element, involving $2n-1$ unknown parameters (comprising $n-1$ random numbers and n gradients for each backward propagation). As n is greater than 1, this function group does not disclose any information regarding the precise values of Δw_i and Δb_i. Instead, it reveals the subspaces that numerous potential matrix solutions can linearly combine. Consequently, the server cannot reconstruct the intermediate gradients.

Theorem 1. According to [1] the weight gradient Δw and bias gradient Δb reveal nothing if r_i^w and $r_i^b = N(0, \frac{\sigma^2 I_{iter}}{B^2})$, where N is the Gaussian distribution with mean 0 and standard deviation $\frac{\sigma I_{iter}}{B}$, I_{iter} is the number of training iterations, σ is the noise level and B is the mini-batch size.

5.4 SYSTEM EVALUATION

5.4.1 Experiment Setup and Data Collection

The test bed consists of a victim IoT device built from Hardkernel Odroid-XU3, with multiple workstations as the smart gateway (client) and the server. Various clients communicate with the server via a local area network (LAN) with a bandwidth of about 1 Gbps. Our power datasets are collected under three different states: Idle (e.g., the IoT device is not running any app or service), IoT routine service (e.g., the device is running an app), and malware infection (e.g., the device is under botnet infection). 12,000 instances were collected for the idle class, 12,000 for the IoT service class, and 17,000 instances for the botnet attack class to train the model. We split the dataset into 80:20 rule, so 80% of the data is used for training and 20% for testing. The experiment tested the following malware, namely Mirai, Mirai variants, LuaBot, Remaiten, IRCBot, Qbot, and Ransomware (MedusaLocker).

Table 5. Classification results of BoTShield model

Class	**Accuracy**	**Precision**	**Recall**	**FPR**
Two-class	98.4	97.2	99.5	2.87
Three-class	97.1	95.4	98.6	4.95

5.4.2 Performance of Secure Online Training

Our model has four hidden layers, namely convolution, ReLU, the fully connected layer, and the soft-max layer. The input layer has a dimension of 1×2550, and the output is a 3-element vector containing the classification probabilities of the three classes (i.e., idle, service, and botnet attack). Hyper-parameter selection and tuning are then conducted to get the best model. Specifically, the model is fine-tuned by evaluating its performance under different kernel sizes (e.g.,

1×32, 1×64, 1×128) and strides (e.g., $1 \times 8, 1 \times 16, 1 \times 32$ of the convolution layer. The number of kernels/filters in the convolution layer is 16.

We remark that this is the first work to show the feasibility of secure training to detect IoT botnet attacks that achieve high levels of accuracy. Our secure training model achieves an average overall accuracy of 97.1% for a three-class classification. Considering that our system targets IoT botnet detection, it can be simplified from a three-class (Idle, IoT service, and Botnet) classifier to a two-class (Botnet and Non-Botnet) classifier to identify whether an IoT device is infected into a bot. For two-class classification, our model achieves an accuracy of about 98.4% and a false positive rate (FPR) of 2.87%. The results are listed in Table 5

To illustrate the necessity of the online model, we want to compare the results of offline and online training models. Initially, we trained an offline model exclusively using the original Mirai infection data and tested it against Mirai variants and ransomware. The primary objective is to

Table 6. Detection of malware variants for offline training method.

Input	**Metric**	**Satori**	**Katana**	**IRCBot**	**QBot**	**Ransomware**
Mirai	Accuracy	77	80	75	78	76
	Recall	68	72	65	80	69
	Precision	80	75	79	72	75

determine if the offline model can detect unknown malware. The results are listed in 6. The average detection accuracy is around 75%, suggesting that the current offline training approach is not effectively addressing the detection of unknown malware. This limitation arises because the training model is trained with original Mirai power data only, leading it to learn unrelated features that do not exist in new malware while attempting to identify potential new variants. But,

Table 7. Detection of malware variants for online training method.

Input	**Metric**	**Satori**	**Katana**	**IRCBot**	**QBot**	**Ransomware**
Mirai	Accuracy	94	97	95	98	93
	Recall	91	97	93	94	96
	Precision	96	92	96	99	93

in our online training model, the new variants data can be fed directly into the model in real-time. Therefore, the online training model performs much better than the offline model. The detection accuracy of online results for various malware is shown in Table 7.

5.4.3 Performance Comparison with State-Of-The-Art Works

We evaluate our protocols for secure training in the LAN settings. In our experiment, multiple clients communicated with the server via LAN; otherwise, we couldn't bring the latency time to under 0.5 seconds. The clients collect the data from the IoT device and then send it to the server. After that, the client communicates back and forth with the server to evaluate the whole model. First, we want to assess our model when only one client communicates with the server.

Table 8. BoTShield execution times for one client-server model

Batch Size	IoT to Client (ms)	Per Sample time (ms)
1	500	2155
8	1000	930
32	3500	650
64	6500	580

Table 8 summarizes the result in LAN settings when the epoch is 25 and batch size varies from 1 to 32. The smart gateway receives the packet every 0.5 seconds from the IoT device; therefore, our model couldn't achieve real-time malware detection for one client-server model. During this client-server interaction, the latency between IoT and the client is vast compared to the detection latency. Thus, we employ multiple clients to reduce the latency time. We vary the batch size between 8 and 64 to evaluate the computation time of BoTShield when six clients simultaneously communicate with the server to support 50 to 100 IoT devices.

Table 9 presents the results when the batch size is varied, and the number of epochs is fixed to 25. Our model detects malware within 0.5 seconds and achieves considerable accuracy when the number of epochs is 25, the batch size is 32, and the number of clients is six. Our design outperforms one client-server detection latency when we employ multiple clients. The reason behind this is that when we use various clients to pack data and send it to the server, the vectorization computation of the model is improved. Millisecond is abbreviated as ms in the following tables.

Now, we compare the computation speed of BoTShield with SecureML [56] and SecureNN [78].

Table 9. BoTShield execution times for 25 epochs and multiple client-server models.

Batch Size	**Accuracy**	**LAN (hours)**	
		Per Sample Time (ms)	Total Time (hrs)
8	98.1	670	6.09
16	97.8	441	4.02
32	97.3	300	2.73
64	96.5	297	2.71

As we've already said, the offline phase is completed before the client receives the training data, so we only consider the online phase time the total time. We compare our protocols with their work in Table 10. In the LAN setting, our protocol is roughly 1.39× faster than SecureNN and 4.43× faster than SecureML 3-server settings.

Table 11 presents the microbenchmarks of our model. Forward propagation (FP) latency time is smaller than backpropagation (BP). We can see from Table 11 that most of the time in our model is spent on the fully connected (FC) and the convolution (ConV) layer of the backpropagation. This is because we must encrypt the loss function operations in these phases to protect data privacy in the training phase. Even though our model couldn't achieve real-time detection for the training phase, it achieved real-time detection during the inference phase. it achieved

The performance of BoTShield protocols is evaluated in the context of secure inference, with Table 12 summarizing the comparison with state-of-the-art secure inference protocols. As depicted in Table 12, the inference model exhibits impressive speedups of up to 550×, 55.55×,

Table 10. BoTShield training time comparison for batch size 32 and 25 epochs with SecureML and SecureNN.

Model	**LAN (hours)**			**Per Sample Time (ms)**
	Offline	Online	Total	
SecureML	5.8	5.5	12.10	1240.24
SecureNN	0	3.8	3.8	417
BoTShield	1.84	0.89	2.73	300

16.67×, 7× and 2.36× when compared to GAZELLE [37], DeepAuditor [35], SecureNN [78]and MUSE [42] respectively, especially when dealing with convolutional layers (CONV) of a kernel size of 1×64 and a stride of 16. Regarding communication cost, the model incurs significantly less transmission load during the online phase, primarily involving plaintext data. Consequently, the proposed inference method reduces online communication costs by more than 250×, 200×, 20×, 6.8×, and 2.5× compared to GAZELLE, DeepAuditor, SecureNN, and MUSE when using a kernel size of 1×64 and a stride of 32. Furthermore, in the fully connected layer (FC), the proposed model demonstrates a substantial advantage in online runtime, achieving speedups of up to 2350×, 1.45×, 8×, and 7× compared to GAZELLE, DeepAuditor, SecureNN, and MUSE, as outlined in Table 12. Our scheme outperforms previous work by several orders of magnitude regarding the runtime. This is mainly because we don't rely on any heavy cryptographic primitives. In addition, generating and transmitting garbled circuits is time-consuming, particularly for such data and computation-intensive tasks; thus, we avoid using garbled circuits in the non-linear layers. It is

Table 11. BoTShield backpropagation microbenchmarks for per sample and 25 epochs in LAN settings.

Batch Size	**FP**	**Backpropagation (ms)**			
		SoftMax	FC	ReLU	ConV
8	25.3	0.6	215.21	0.05	448.85
16	12.50	0.6	149.60	0.05	289.25
32	7.20	0.5	106.80	0.05	185.30
64	7.01	0.5	105.80	0.05	184.64

evident from the above experiments that the state-of-the-art listed in Section 5.4 fails to detect IoT botnet attacks in real-time, whereas BoTShield can detect IoT botnets with high accuracy because of its advantageous design.

Table 12. Per sample inference time (ms) and communication cost (MB) comparison of various protocols

Layer	**Method**	**Time (ms)**		**Communication (mb)**	
		Offline	Online	Offline	Online
Conv	GAZELLE	—	3956	—	8.033
	DeepAuditor	0	400	0	15
	SecureNN	0	120	0	9.1
	Muse	2600	55	15	1.3
	BoTShield	350	7.2	8.65	0.032
FC	GAZELLE	—	141	—	0.503
	DeepAuditor	0	120	0	2
	SecureNN	0	1.2	0	0.012
	Muse	220	0.41	4.5	0.025
	BoTShield	60	0.6	1.38	0.003

CHAPTER 6

MALWARESHIELD: A PRIVACY-PRESERVED FEDERATED LEARNING FRAMEWORK FOR ZERO-DAY MALWARE DETECTION ON IOT DEVICES

6.1 SUMMARY

With the Internet of Things (IoT) becoming a fundamental aspect of our daily routines, these devices bring considerable convenience and efficiency but also introduce unforeseen risks and emerging threats. IoT devices are particularly attractive to cybercriminals due to their constrained resources, continuous internet connectivity, and lack of robust security features. Recently, numerous high-profile attacks on IoT devices have been reported [2], [3]. These compromised devices can be used to execute Distributed Denial of Service (DDoS) attacks [2] or Permanent Denial of Service (PDoS) attacks. Botnet attacks pose a significant cybersecurity threat to the Internet of Things (IoT) networks [8], [60]. A botnet is a network comprising numerous malware-infected devices which can be used to launch cyber-attacks from within. For example 2016, a massive DDoS attack knocked down the DNS service Dyn. It made several Internet platforms and services, such as Amazon, Netflix, PayPal, and Twitter, temporarily unreachable to numerous users in Europe and North America, which Mirai Botnet caused. Since then, new variants are evolving and exploring a variety of vulnerabilities in unsecured IoT devices. In May 2023, a new Mirai variant named HinataBot was identified, requiring less than 1% of Mirai's resources yet generating substantial traffic comparable to Mirai's most aggressive attacks [4]. Since then, new variants are evolving and exploring a variety of vulnerabilities in unsecured IoT devices. In 2021, ransomware attacks were reported every 11 seconds, with projections indicating this frequency could accelerate to every 2 seconds by 2031 [15]. Current research categorizes IoT malware attacks into three stages: intrusion, infection, and monetization [57]. To minimize the loss caused by IoT malware, identify these malicious activities as early as possible, e.g., the intrusion stage.

Malware detection methods typically fall into two main categories: network-based and host-based approaches. Network traffic analysis [29], [55] is frequently employed to safeguard IoT systems. Detecting botnet attacks on IoT devices remains challenging despite clear intrusion procedures, primarily due to the subtle nature of malicious network traffic in the early stages of an attack. Deploying network-based botnet detection systems across various IoT devices and vendors is heavyweight and intrusive to users in this scenario. An alternative approach is to use host-based security mechanisms [73] to detect these activities. However, IoT devices are typically resource-constrained, making it challenging to support comprehensive detection solutions. Moreover, these devices are designed based on diverse principles and mechanisms, complicating applying a single, universal detection approach across all device types. A practical approach to addressing this issue involves leveraging power side-channel information to detect malware attacks. This method is robust and versatile because power traces are difficult to compromise and can capture aggregated activities across diverse devices, including various hardware, vendors, and operating systems. Moreover, adversaries find it nearly impossible to mimic normal power consumption behavior during attacks. Early work in this field utilized power side-channel data to detect malicious behavior on mobile devices in the early 2010s [40], [82]. Still, these studies require validation for application in IoT environments today.

Recent research has explored the correlation between malicious activities and side-channel effects, i.e., power consumption [19], [35], [45], [46]. These studies have predominantly focused on offline or online training with limited functionality. Thus, there are more suitable approaches for detecting new malware variants. Some researchers proposed detection models to detect botnets on IoT devices through Federated Learning (FL) [60], [61]. While these existing distributed approaches have successfully constructed botnet attack detectors, they still need to be more optimal in effectively addressing zero-day malware attacks. Detecting these attacks in IoT networks presents challenges due to the absence of prior knowledge regarding such malicious incidents across most IoT devices. But, with the rapid expansion of IoT applications, timely detection of such attacks within the IoT domain is crucial. Therefore, this work is primarily motivated by the need to develop an online training malware detection strategy capable of keeping up with the rapid pace of malware evolution.

Federated Learning (FL) is a popular paradigm that can detect malware attacks on IoT devices. Despite this, federated learning still poses privacy risks. For instance, research has demonstrated that servers (aggregators) can potentially recover private information about target users by analyzing their uploaded local models [25], [87]. However, user data from IoT devices is privacy-sensitive, and model parameters at the server are considered proprietary information. So, the FL framework can leak privacy-sensitive information. To address such security issues, we propose MalwareShield, a privacy-preserving federated learning (PPFL) framework that can detect zero-day malware attacks on IoT devices. Our MalwareShield framework has an autoencoder model to learn complex patterns, e.g., of various IoT device states. We extract statistical features that capture behavioral snapshots of benign IoT traffic and train individual deep autoencoders for each device to learn normal IoT behaviors. These autoencoders aim to compress snapshots, and failure to accurately reconstruct a snapshot strongly indicates malware attack behavior. We only train MalwareShield with benign data, and no malware data is seen during the training phase, only in the test phase. Therefore, it perfectly simulates real-world zero-day malware attack scenarios.

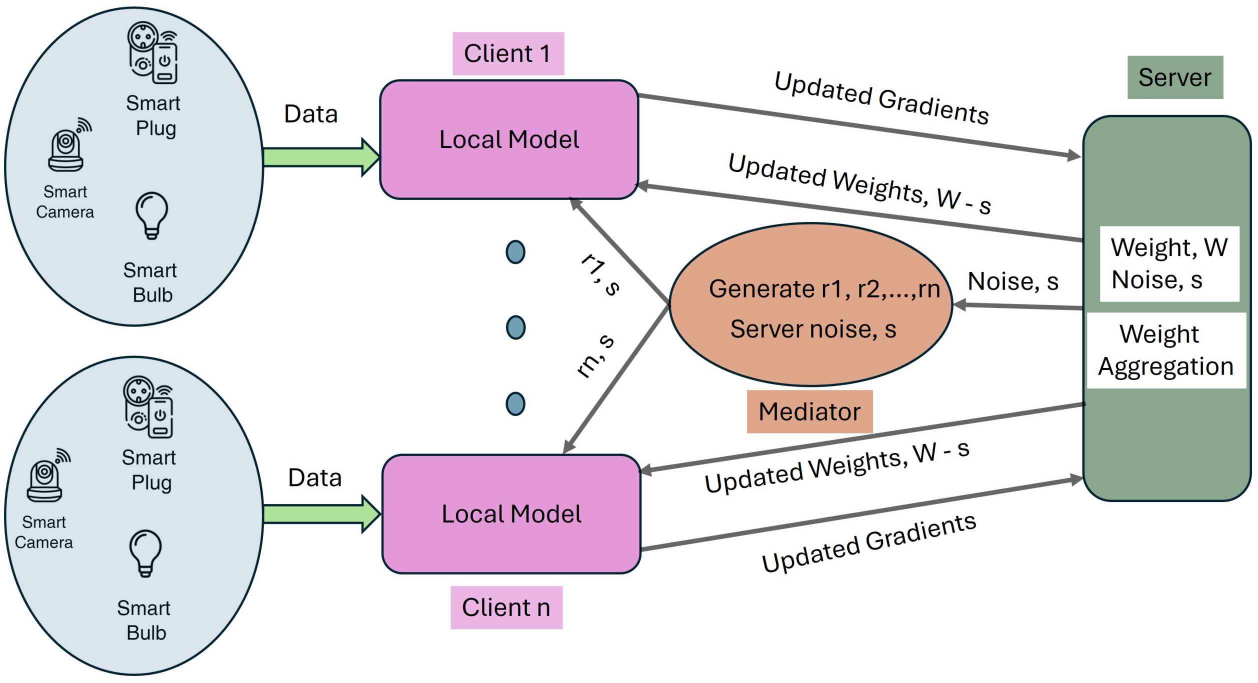


Figure 15. System design of MalwareShield.

6.2 SYSTEM DESIGN OF MALWARESHIELD

To detect zero-day malware infections accurately, we have developed an IoT malware detection system called MalwareShield, which monitors power side-channel signals. This system utilizes current sensors, data acquisition, and preprocessing techniques to extract malicious signals effectively. Our objective is to perform a detailed analysis of how malware execution leaves detectable traces in the power data of IoT devices. We aim to experiment with deep learning techniques to evaluate their effectiveness and efficiency in detection. MalwareShield consists of IoT devices, a mediator, multiple clients for data collection, preprocessing, and local training, along with an innovative privacy-preserving federated learning module for the client and server (aggregator), as illustrated in Figure 15. In this section, we outline the system architecture of MalwareShield and discuss the methods used to generate power datasets for effective feature extraction.

6.2.1 System Overview

MalwareShield depicted in Figure 15 comprises IoT devices, multiple clients, and an aggregator. A low-cost current sensor is integrated into the device to monitor an IoT device's power consumption continuously. This sensor samples the direct current (DC) power supplied to the device under two states (i.e., benign and attack). Through an in-depth analysis of the malware infection process, we discovered that abnormal power spikes occur when Linux commands are executed on the device during the scanning and post-processing phases. Multiple IoT devices are connected to a single client, each with its local model. Each IoT device sends its power data to the client in a separate channel. We deploy an autoencoder model on the client side, while the aggregator has the initial model parameters. The mediator is responsible for generating noises $(r_1, r_2, ...r_n)$ and providing them to the client to ensure the privacy of the client's data. The mediator also receives noise (s) from the server and sends it to the client at the beginning of each iteration.

6.2.2 Datset Generation and IoT-Client Interaction

The critical aspect of dataset generation is collecting and preprocessing suspicious power signals to extract distinguishable features. The client is engineered to streamline efficient data collection, preprocessing, and local training. During data collection, the power data is sampled by the on-device current sensor and live-streamed to the client for preprocessing. As multiple IoT devices are connected to a single client, each client is designed to efficiently and practically collect data through multi-thread data streaming. The collected power side-channel data is segmented using an appropriate sliding window size and then placed into a local queue, awaiting retrieval as an instance for the training module. The malware infection period, including brute-force scanning and post-processing, is shorter than 0.3 seconds. Thus, the sliding window is set to 0.3 seconds. In our prototype system, the sampling rate of the integrated current sensor is 1 kHz, and the size of the sliding window is 0.3 seconds, resulting in an input instance size of 1×300.

6.2.3 Overview of Client-Mediator-Aggregator Interaction

The bottleneck of the privacy-preserving federated learning approach is the highly costly computation time and accuracy loss. Therefore, we avoid using an encryption-based federated learning approach to build the framework. We use a differential privacy-based approach to build MalwareShield. We adopt a noise removal approach in the aggregator to reduce the accuracy loss. Each client has its own local model to ensure the privacy of the client's data. Despite this, federated learning still poses privacy risks. Research has shown that aggregators can potentially recover private information about target users by analyzing their uploaded local models [25], [87]. Therefore, we introduce a mediator entity in our framework. The mediator's purpose is to generate random noises and send these noises to the client to mask the model parameters (weight gradients). When the train finishes for the current iteration, each client will send their updated model to the aggregator. The aggregator cannot recover private information from the client's data as the noises are added to the updated model parameters. However, the noises accumulate in the aggregator, so removing the noises before the start of the next iteration is essential. We will discuss the noise removal approach in Subsection 6.3.3. The purpose of the aggregator is to provide the initial model parameters to the client. Later, when it receives the updated model parameter from the client, it aggregates all the model parameters to initiate the next iteration. We also inject the random noise (s) into the model parameter on the aggregator side so that the attacker cannot infer the model parameters. The privacy requirement of MalwareShield is that it prevents the aggregator and attacker from accessing the client's private data. Moreover, it also prevents attackers from inferring model parameters.

6.3 DESIGN OF MALWARESHIELD FRAMEWORK

This section discusses the MalwareShield framework for multiparty semi-honest settings. Subsections cover malware detector training on the client side, the internal structure of the autoencoder model, and secure federated learning.

6.3.1 Malware Detector Training on the Client Side

Our malware detector employs deep autoencoders, each client maintaining a separate model and connecting multiple IoT devices to a single client. An autoencoder is a neural network designed to reconstruct its inputs after compressing them, which helps it learn meaningful patterns and relationships among input features. When trained exclusively on benign instances, an autoencoder will effectively reconstruct normal observations but struggle with malware instances (unknown concepts). Significant reconstruction errors thus signal potential malware attacks, allowing us to classify such observations accordingly.

We optimize the hyperparameters of each trained model to maximize the true positive rate (TPR) and minimize the false positive rate (FPR) when evaluating unseen traffic. We use two datasets to enable the model to learn normal activity patterns. The training set (D_{train}) trains the autoencoder, adjusting input parameters such as the learning rate and the number of epochs. We train the model using only the normal data. We use the attack data as validation and normal data to generate reconstruction errors for both the normal and attack data. The validation set (D_{val}) is used to iteratively refine hyperparameters such as the learning rate and number of epochs until the mean squared error (MSE) between the model's input (original feature vector) and output (reconstructed feature vector) stabilizes. This process helps prevent overfitting and improves detection accuracy with new data. The validation data set is further employed to optimize a threshold (tr) for distinguishing between benign and malicious observations to minimize FPR. Once the model training and optimization are complete, the normal threshold (tr) is determined. To determine this threshold, we select a value of one standard deviation above the mean reconstruction error. This approach provides a balanced measure of detection sensitivity and specificity. This threshold, above which an instance is classified as malware, is computed as the sum of the sample mean (μ) and standard deviation (σ) of MSE across the validation set (refer to Equation2).

$$tr = \mu(MSE_{D_{val}} + \sigma(MSE_{D_{val}}) \tag{2}$$

6.3.2 The Structure of Autoencoder

An autoencoder network is a specialized, unsupervised neural network within deep learning. It consists of two primary components: an encoding network and a decoding network, as shown in Figure 16.and decoding networks. The encoding network compresses and reduces the dimensionality of the input data, while the decoding network reconstructs the original input from this compressed representation. The network's loss function quantifies the error between the original input and the reconstructed output. Training the autoencoder involves minimizing this loss function through iterative gradient updates, which defines the network's operational mechanism.

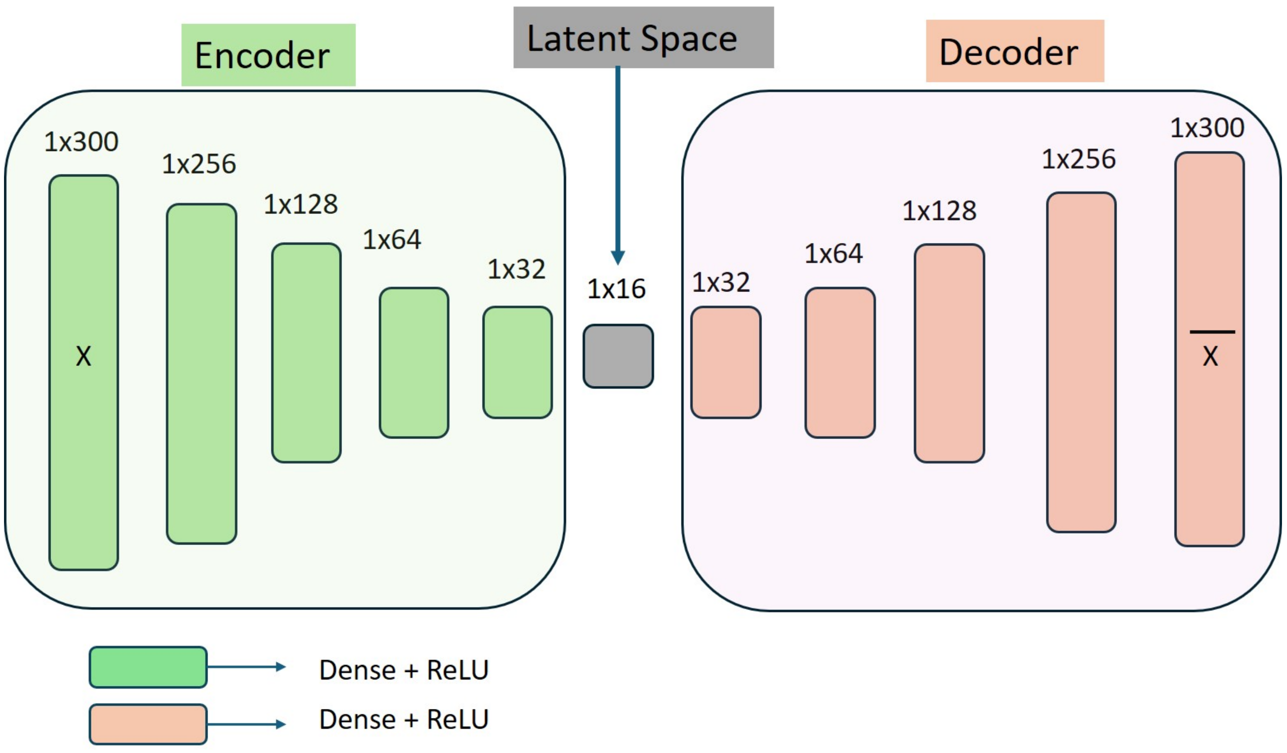


Figure 16. Internal architecture of autoencoder Model.

The autoencoder model is designed to be used for malware detection. In an autoencoder, encoding refers to transforming the input layer to the hidden layer, while decoding involves changing from the hidden layer to the output layer. The encoder takes the input feature vector and uses ReLU activation functions in the hidden layers to generate new features. The first layer of this model is the input layer, and the dimensions of this layer are 1×300. The next layer is the dense layer, followed by a non-linear ReLU layer. The number of neurons in this layer is 256. The total trainable parameters of this hidden layer are $300 \times 256 = 76800$. The next dense layer contains only 128 neurons. Therefore, during the encoding phase, our model downsamples the input to a smaller dimension until it reaches a dimension 1×16. The input to the decoder model is 1×16. Then, it upsampled the input until it reached an exact dimension of the input layer, 1 $times$300. The decoder output is denoted as $\bar{x}$. $\bar{x}$ is an estimated value our model generates. During the first phase, feed-forward propagation is performed for each input x to obtain the estimated output value $\bar{x}$. The error of our model is |x - $\bar{x}$|. In the second phase, the error will be backpropagated through the network to revise the weights/neurons. In the adjusting phase, we tweak the hyperparameters at each layer to get the best results from the model. We found during the experiment that if we use the number of neurons as depicted in Figure 16, our model produces the best result.

6.3.3 Secure Federated Learning

Our secure Federated Learning (FL) system consists of one server, mediator, and N clients, as depicted in Figure 17. Let D_j denote the local database held by the client C_j, where $j \in 1,2,,...,N$. At the server, the goal is to learn a model based on data that resides at the N associated clients. An active client participating in the local training must find a vector W_i, where $i \in 1,2,...,M$ of the autoencoder model to minimize a certain loss function. M denotes the maximum number of iterations/communication rounds the model needs to reach the optimal value. Formally, the server aggregates the weights sent from the N clients as

$$W_{i+1} = \sum_{i=0}^{N} k_i W_i, \quad W_i = W_i - \eta \Delta W_i \tag{3}$$

where w_i is the weight trained at the i-th client, w is the weight after aggregating at the server, Δw_i is the weight gradient, η is the learning rate, N is the number of clients, $k_i = \frac{|D_i|}{|D|} \geq 0$ with $\sum_{i=0}^{N} k_i = 1$ and $|D| = \sum_{i=0}^{N} |D_i|$ is the total size of all data samples.

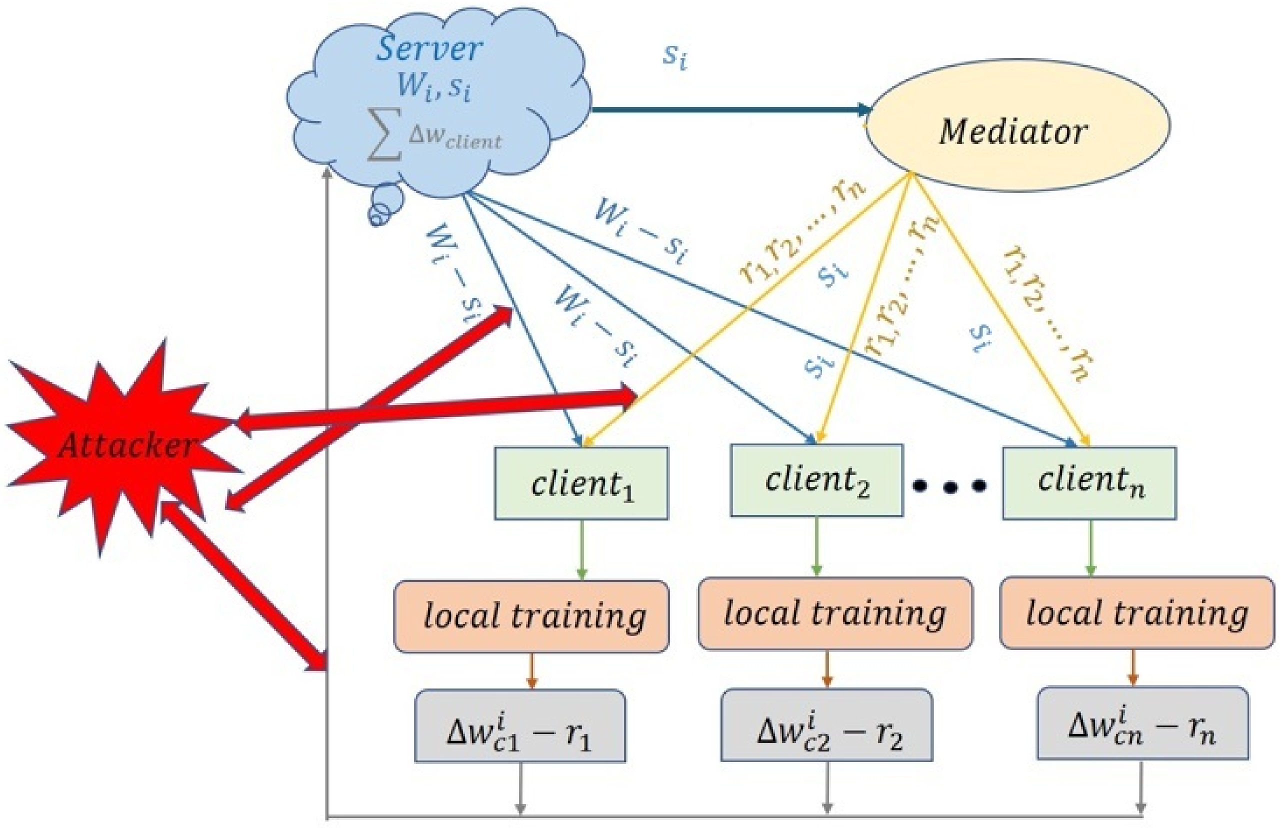


Figure 17. Secure federated learning.

If we do the aggregation as described in Equation 3, the researcher has demonstrated that servers (aggregators) can potentially recover private information about target users through analysis of their uploaded local models [25], [87]. Therefore, the client cannot send the original weight gradients Δw to the aggregator. One way to hide the gradient from the server is to mask the gradient before sending it to the aggregator. But, if all the clients mask the gradient, the error will accumulate in the aggregator during aggregation. As a result, the accuracy of our model will be poor. To resolve this issue, we introduce a Mediator in our framework. The Mediator will be involved during the training phase of MalwareShield, but it serves no purpose during the test phase. In each iteration of the training phase, the mediator first calculates the number of clients participating in collaborative training. Suppose there are N active clients during that particular iteration. Then, the mediator generates N random noises to satisfy the following equation.

$$\sum_{j=0}^{N} r_j = 0 \tag{4}$$

In Equation 4, r_j is the random noise generated for the j-th client. Then the mediator sends the noises to all the clients, e.g., $client_1$ receives noise r_1. Each client has its autoencoder model. The mediator sends the random noises to the client after the client starts training to improve the computation time. The training process takes longer than the communication time from the mediator to the client; therefore, we can hide the communication latency if we send a random noise after the client enters the training phase. After the training iteration, the updated weight gradient is Δw_{cj}^{i} for the j-th client. Then the j-th client masks its weight gradient during i-th iteration as follows:

$$\Delta w_{cj}^{i} - r_j \tag{5}$$

After each iteration, the client sends its updated gradient to the server. If any attacker probes the communication channel, it only gets $\Delta w_{cj}^{i} - r_j$ but not the original gradient. Hence, by masking the gradient, we can protect it from the attacker. After the server receives the gradient from all the clients, it will aggregate it to update the weight for the next iteration. In this case, the server gets the $\Delta w_{cj}^{i} - r_j$ from j-th client; therefore, the aggregator cannot recover private information about the target client by analyzing their uploaded local models. The aggregated weight is calculated as follows:

$$W_{i+1} = W_i - \eta \Delta w_i, \quad \Delta w_i = \frac{\sum_{j=0}^{N} \Delta w_j^i - r_j}{N} \tag{6}$$

From Equation 2 we know that $\sum_{j=0}^{N} r_j = 0$. Therefore, when the server does the aggregation, all the noises are canceled out, and the only thing the server gets is the aggregated weight gradient

Δw. Then, the server again sends the updated weight to the client to initiate the next iteration. The server also masks the updated weight as follows:

$$\Delta W_i - s_i \tag{7}$$

where s_i is the random noise generated by the server. The reason behind this is that the attacker may probe the communication channel between the server and the client; therefore, the masked weight protects the privacy of server parameters. After the client receives the updated weights, the mediator gets the random noises r_i and s_i. Before the client enters the training phase, the noises from the weight $W_i - s_i$ are removed by adding s_i. If we don't remove the noise from the weight, the performance of our model will be degraded. To improve the privacy of our framework, the mediator generates random noises in each iteration and sends these to the client after the client starts the training process.

Theorem 1. According to [1] the weight gradient Δw_i and bias gradient Δb_i reveal nothing if $r_i = N(0, \frac{\sigma^2 I_{iter}}{B^2})$, where N is the Gaussian distribution with mean 0 and standard deviation $\frac{\sigma I_{iter}}{B}$, I_{iter} is the number of training iterations, σ is the noise level and B is the mini-batch size.

6.4 SYSTEM EVALUATION

We evaluate the classification accuracy, including precision and recall, to validate MalwareShield's detection performance. We present MalwareShield's detection accuracy and microbenchmark based on the power data from the IoT devices (i.e., Raspberry Pi 3 model-based security camera). We compare the detection performance with state-of-the-art privacy-preserving systems such as DeepAuditor [35], ThingNet[46] and DeepShield[39].

6.4.1 Data Collection

To validate the robustness of MalwareShield, we collect power datasets under two different states: Normal (i.e., the IoT device is not running any app or the device is running security camera services) and Attack (i.e., the device is under malware attack). The model takes the input of power instances and classifies them into one of these classes. We have collected around 20000 samples of benign data and 22000 samples of attack data to feed into the model. We split the dataset into 80:20 rule, so 80% of the data is used for training and 20% for testing. Only the benign data is used during the training phase, but both benign and attack data are used during the validation and testing phase.

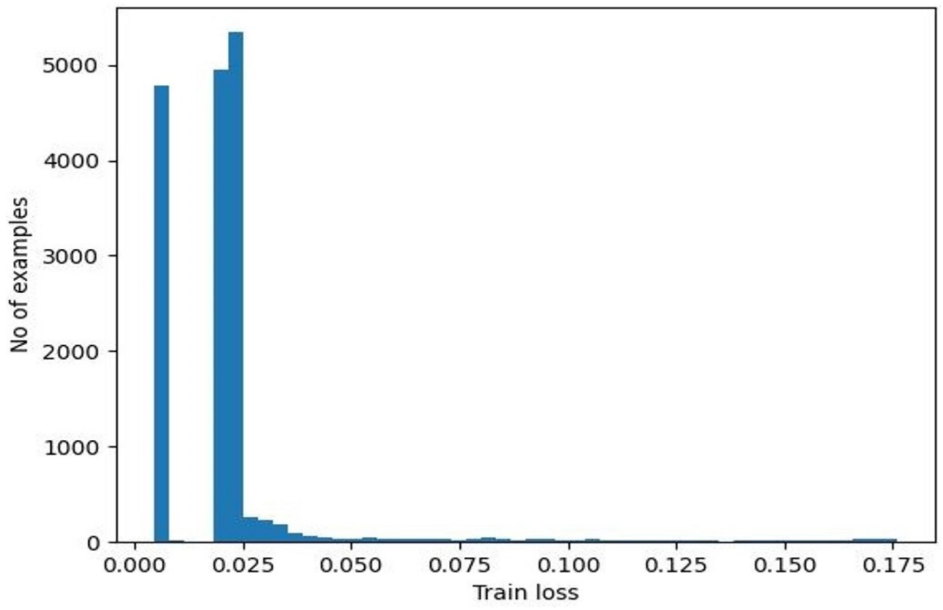


Figure 18. Train loss for benign data.

6.4.2 Performance of MalwareShield

The training objective for the MalwareShield framework is to minimize the reconstruction error—the difference between the input data and the reconstructed output. Suppose we train the model on benign data. In that case, it learns a reconstruction function that works well for normal-looking data (low reconstruction error) and poorly for malware data (high reconstruction error). We can then use reconstruction error as a signal for malware detection. In Figure 18, the train loss/error for benign samples from the training set is plotted on the X-axis, and the number of training samples is plotted on the Y-axis. If we set the threshold value for benign data to 0.025, most benign samples will be classified accurately. In Figure 19, the test loss/error for malware samples

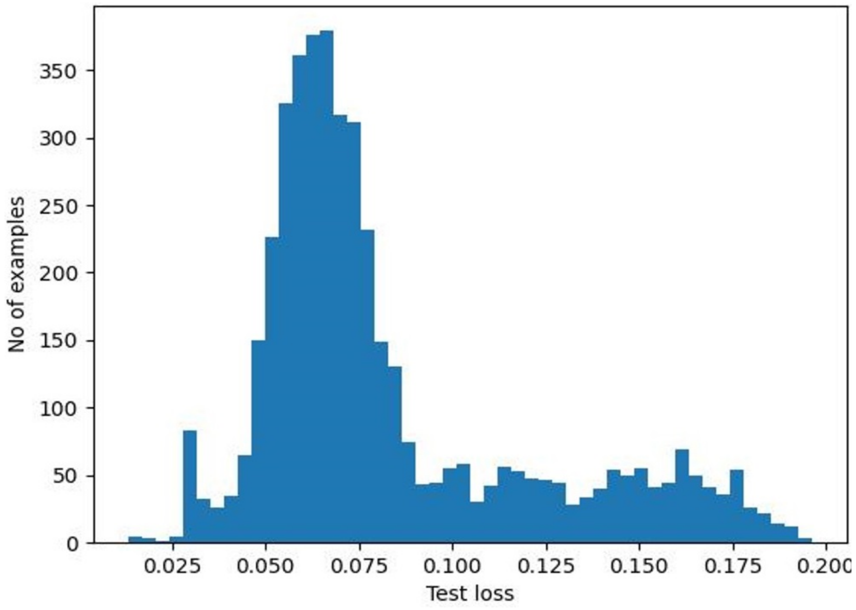


Figure 19. Test loss for malware data.

from the test set is plotted on the X-axis, and the number of test samples is plotted on the Y-axis.

In this example, most samples' test loss is over 0.025. Some sample errors are below 0.025, so those samples are falsely classified as benign data. But we can see from the figure that this number is minimal. So, the false positive of our model is very low. If we go back to figure 18, we can see that some of the sample's errors are over 0.025. Therefore, those samples are falsely classified as malware data. However, this number is also tiny compared to the total number of samples. The false negative of our model is also very low. In our next experiment, we compared our model with

Table 13. Classification performance comparison of DeepShield, DeepAuditor, ThingNet, and MalwareShield.

	Accuracy	Precision	Recall
DeepShield	58.20	63.12	65.30
DeepAuditor	60.16	66.15	60.34
ThingNet	65.28	72.60	59.69
MalwareShield	95.16	95.63	94.19

DeepShield, DeepAuditor, and ThingNet. The results in terms of accuracy, precision, and recall are summarized in Table 13 compared with DeepShield, DeepAuditor, and ThingNet. From the table, it is evident that MalwareShield outperforms all the state-of-the-art models. Figure 20 presents the confusion matrix for our test sets. True label '0' represents attack data, and '1' represents benign data. Out of 4480 samples of attack data, 4301 samples are classified correctly. For the attack

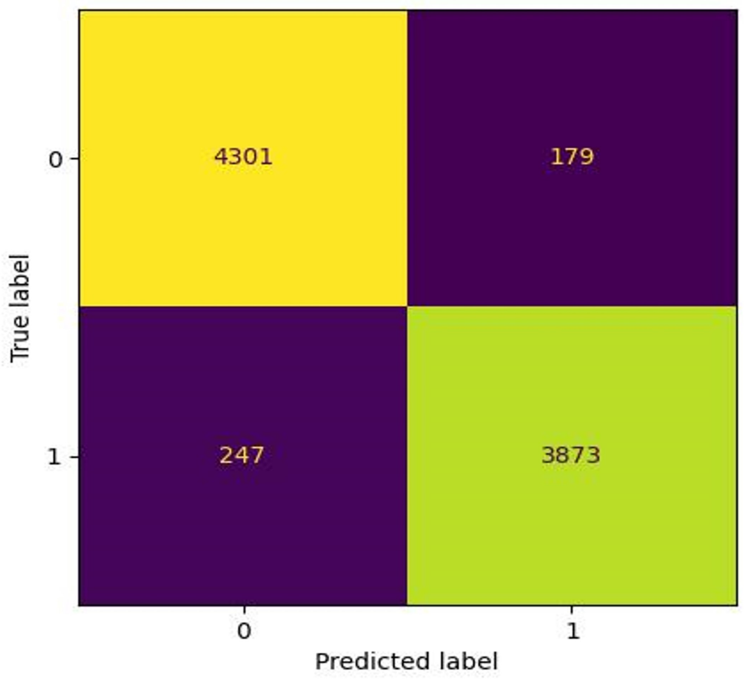


Figure 20. Confusion matrix for test dataset

class, the false positive rate is around 3.99%. On the other hand, out of 4120 samples of benign data, 3873 samples are classified correctly. Therefore, the false negative rate for the benign class is around 5.99%.

In federated learning, the classification accuracy of an intermediate model is affected by both the number of data points each client has and the total number of clients involved. To evaluate the impact of these factors, we record our framework's classification accuracy and training time across different numbers of clients and varying amounts of data points per user. Here, we utilize the power dataset and use the symbols |C| and |D| to represent the number of clients and data points per client, respectively. We experimented with |C| = 10 clients, each randomly assigned |D| = 500, 1000, or 2000 data points from the power dataset. Figure 21 and Figure 22 presents the classification accuracy and training time for different numbers of data points per user, where a

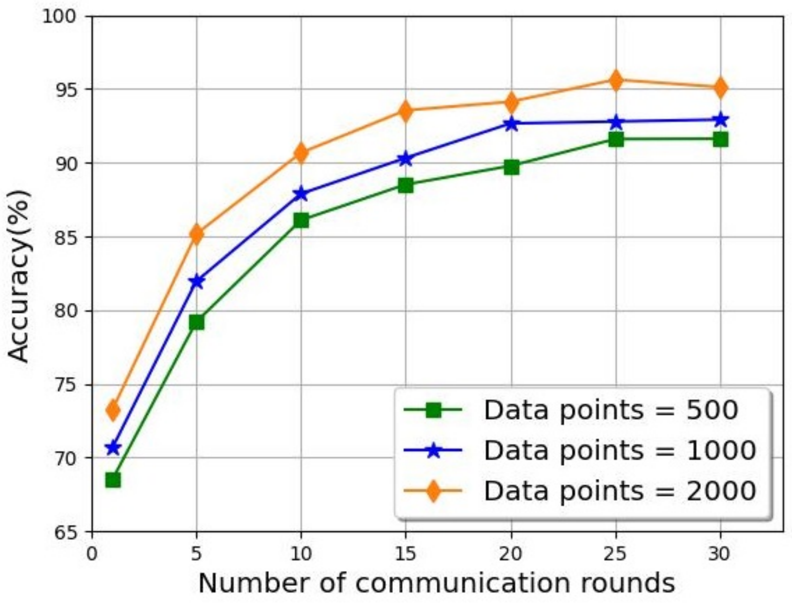


Figure 21. |C| = 10, classification accuracy with different data points per client.

communication round refers to an update of the intermediate model between the aggregator and the clients. In Figure 21 and Figure 22, we observed that increasing the number of data points per client improves the accuracy of the intermediate model and also extends the training time. However, when the number of data points reaches certain thresholds (e.g., 1000 or 2000), the accuracy gains become marginal while the training time increases. Therefore, in future experiments, it is important to select an optimal number of data points per client to avoid unnecessary computational overhead.

In a follow-up experiment, we investigated the effects of varying the number of clients on classification accuracy and training time. We conducted this experiment with three different configurations: |C| = 6, 8, and 10 clients. Each client was randomly assigned |D| = 2000 data points from the dataset. Figure 23 and Figure 24 present the classification accuracy and the training time for different numbers of clients. The results indicate that increasing the number of clients leads

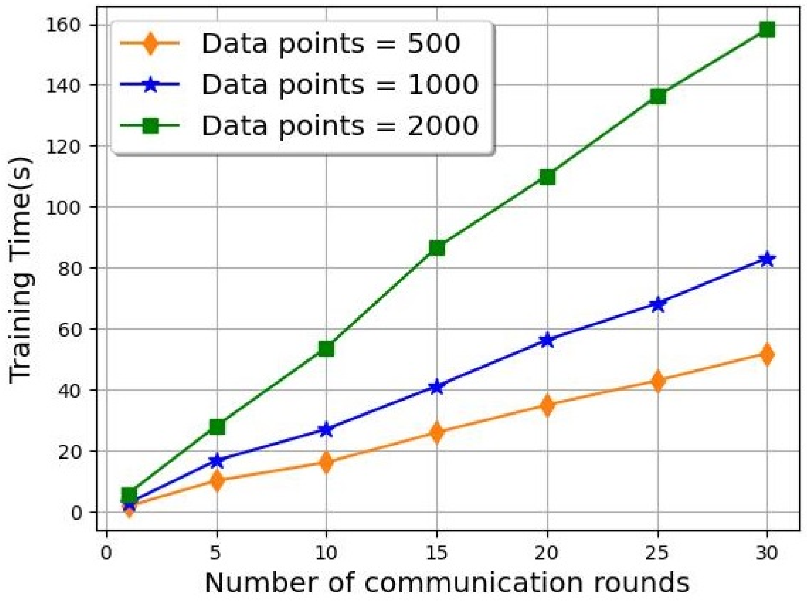


Figure 22. |C| = 10, training time with the different number of data points per client.

to improvements in the accuracy of the intermediate model. This improvement can be attributed to the broader data distribution and diversity among clients, which enhances the model's learning capability. However, this increase in the number of clients also results in additional training time due to the increased volume of data and communication overhead. In Figure 23 and Figure 24, we observe that while the accuracy benefits from having more clients, the training time also rises. The detailed breakdown in Figure 21 and Figure 23, shows that the intermediate model converges after approximately 25 rounds of training. This convergence point signifies that the model has reached a satisfactory level of performance within this number of rounds.

Based on these observations, we decided to perform further experiments using the power dataset. We will record the training time after 25 rounds, capturing data from both the aggregator and the client sides. This approach will provide a comprehensive understanding of how training time and accuracy are impacted by the number of clients and data points in a federated learning

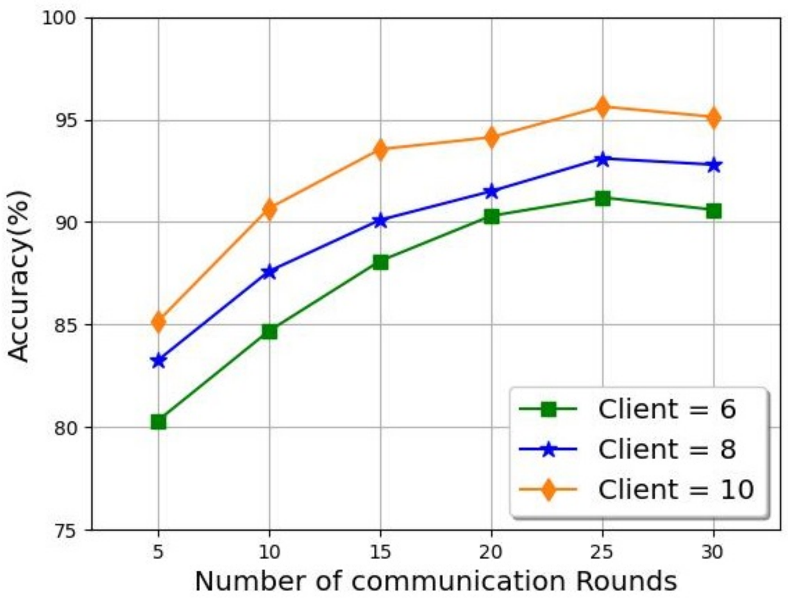


Figure 23. |D| = 2000, classification accuracy with varying numbers of clients.

setup.

In a federated learning setup, clients process their local datasets and transmit updates to the central aggregator, usually as gradients or model parameters. Thus, the aggregator's computational load is more closely linked to the aggregation of these parameters rather than the size of the clients' datasets. As a result, the aggregator's workload remains relatively stable, provided the number of parameters remains constant, regardless of the varying amounts of data each client manages. The analysis of the training process is presented in two key figures: Figure 25 and Figure 26.

Figure 25 illustrates the cumulative training time of the aggregator over 25 rounds of training, considering different numbers of data points per client. Interestingly, the training time for the aggregator remains almost constant, regardless of the increase in data points per client. This phenomenon can be attributed to the fact that the aggregator's training time is primarily influenced by the number of parameters uploaded in each round, rather than the volume of data held by each

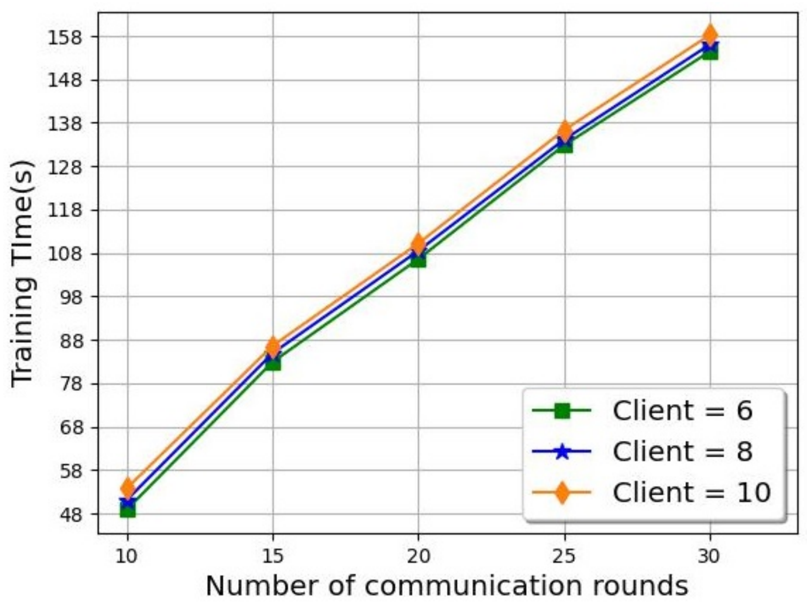


Figure 24. |D| = 2000, training time with varying numbers of clients.

client.

Figure 26, on the other hand, focuses on the total training time for each client after 25 rounds, again varying with the number of data points per client. In contrast to the aggregator, the training time for individual clients shows a linear increase as the number of data points per client rises. This linear trend occurs because the local training time for each client, which significantly impacts the total training time, expands proportionally with the increase in data points. The more data points a client has, the longer it takes to complete the local training phase, leading to a direct and linear escalation in the total training time for each client.

In summary, while the aggregator's training time is largely unaffected by the number of data points due to its dependence on parameter uploads, the clients experience a linear growth in training time as their data volume increases, driven by the demands of local training.

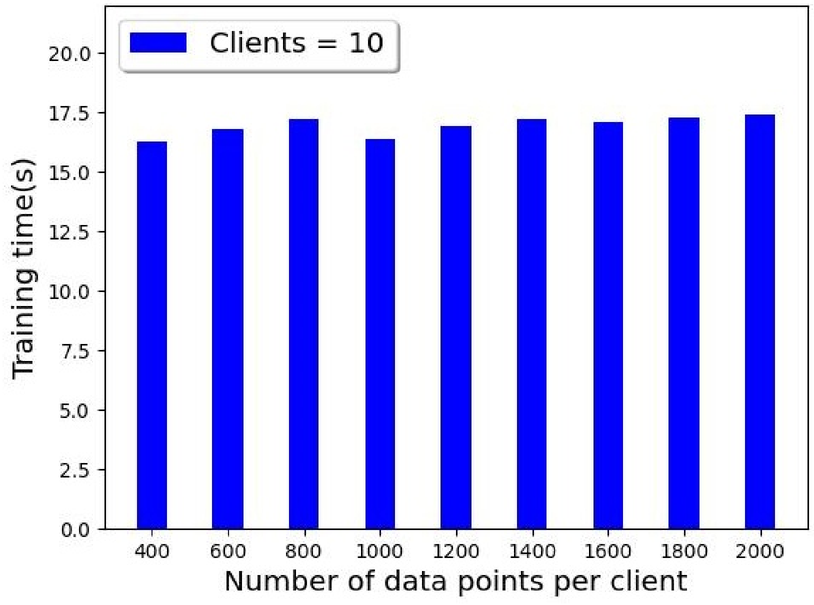


Figure 25. Total training time of aggregator after 25 rounds when |C| = 10, for varying numbers of data points per client.

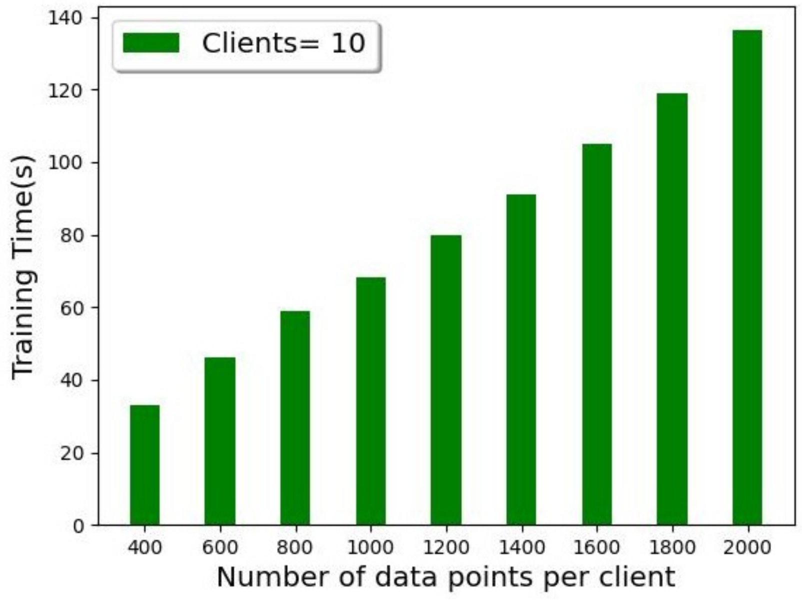


Figure 26. Total training time of each client after 25 rounds when |C| = 10, for varying numbers of data points per client.

CHAPTER 7

CONCLUSION

We investigate the complex correlation between data on power usage and the frequency of malware assaults in the Internet of Things (IoT) ecosystem. Based on an intricate knowledge of the dynamics of IoT malware infection, we carefully design a multi-pronged strategy to improve intrusion detection performance while preserving user privacy.

Our research centers on developing multiple novel deep learning-based intrusion detection systems. The strength of power side-channel analysis is carefully harnessed by these technologies, making it possible to identify possible IoT malware infections. Our approach makes it easier to identify botnet attacks in the early stages and performs better than traditional network-based models, which frequently have to catch up when it comes to early intrusion detection.

At the core of our effort is creating DeepShield, a groundbreaking framework that ushers in a new era in edge computing-based real-time privacy-preserving feature extraction and categorization. DeepShield is an online Convolutional Neural Network (CNN) model that is lightweight and carefully designed to enable quick and precise identification of Internet of Things (IoT) malware trespassing. An innovative hybrid cryptography protocol that balances processing workloads between the edge and the central server is essential to its effectiveness. A smooth and effective functioning is ensured by this thoughtful resource allocation, which greatly reduces computing overhead.

We introduce BoTShield, an advanced privacy-preserving online training methodology in response to the ever-evolving landscape of malware threats. BoTShield represents a significant leap forward by integrating cutting-edge techniques such as homomorphic encryption, secret sharing, and differential privacy. By leveraging these state-of-the-art methodologies, BoTShield enhances privacy and exhibits a remarkable capability to detect diverse malware variants. However, acknowledging its supervised nature, we recognize BoTShield's inherent limitations in preemptively identifying zero-day attacks.

To address this critical gap, we unveil MalwareShield, a groundbreaking federated learning framework fortified by a novel approach to differential privacy. MalwareShield is uniquely equipped with an encoder-based unsupervised model designed explicitly to detect zero-day malware incursions. Through meticulous theoretical analysis and empirical experimentation, we substantiate our frameworks' efficacy, security, and scalability in facilitating secure, real-time, and accurate malware detection within the IoT ecosystem.

REFERENCES

[1] M. Abadi and et. al, “Deep learning with differential privacy,” *Proc. of ACM SIGSAC Conf. on Computer and Communications Security*, Oct. 2016.

[2] K. Angrishi, “Turning internet of things (IoT) into internet of vulnerabilities (IoV): IoT botnets,” *CoRR*, vol. abs/1702.03681, 2017. arXiv: `1702.03681`.

[3] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, *et al.*, “Understanding the mirai botnet,” in *26th USENIX Security Symp.*, Aug. 2017, pp. 1093–1110.

[4] D. Baek, *Around 560,000 malware are generated every day*, 2023. [Online]. Available: `https://www.linkedin.com/pulse/around-560000-malware-generated-every-day-david-sehyeon-baek--i0h1f`.

[5] M. Ball, B. Carmer, T. Malkin, M. Rosulek, and N. Schimanski, “Garbled neural networks are practical,” *IACR Cryptol. ePrint Arch.*, vol. 2019, p. 338, 2019. [Online]. Available: `https://api.semanticscholar.org/CorpusID:115197351`.

[6] E. Bertino and N. Islam, “Botnets and internet of things security,” *Computer*, vol. 50, no. 02, pp. 76–79, Feb. 2017.

[7] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, “Practical secure aggregation for privacy-preserving machine learning,” in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS ’17, Dallas, Texas, USA: Association for Computing Machinery, 2017,

pp. 1175–1191, ISBN: 9781450349468. DOI: 10.1145/3133956.3133982. [Online]. Available: https://doi.org/10.1145/3133956.3133982.

[8] T. M. Booij, I. Chiscop, E. Meeuwissen, N. Moustafa, and F. T. H. d. Hartog, "ToN_IoT: The role of heterogeneity and the need for standardization of features and attack types in IoT network intrusion data sets," *IEEE Internet of Things Journal*, vol. 9, no. 1, pp. 485–496, 2022. DOI: 10.1109/JIOT.2021.3085194.

[9] D. Breitenbacher, I. Homoliak, Y. L. Aung, N. O. Tippenhauer, and Y. Elovici, "HADES-IoT: A practical host-based anomaly detection system for IoT devices," in *Proc. of ACM Asia Conf. on Computer and Communications Security*, 2019, pp. 479–484.

[10] S. L. Campbell and C. D. Meyer, "Generalized inverse of linear transformations," *SIAM*, 2009.

[11] H. Chabanne, A. de Wargny, J. Milgram, C. Morel, and E. Prouff, "Privacy-preserving classification on deep neural network," *IACR Cryptol. ePrint Arch.*, vol. 2017, p. 35, 2017. [Online]. Available: https://api.semanticscholar.org/CorpusID:42011253.

[12] Y. Chang, K. Zhang, J. Gong, and H. Qian, "Privacy-preserving federated learning via functional encryption, revisited," *IEEE Transactions on Information Forensics and Security*, vol. 18, pp. 1855–1869, 2023. DOI: 10.1109/TIFS.2023.3255171.

[13] S. S. Clark, B. Ransford, A. Rahmati, S. Guineau, J. M. Sorber, W. Xu, and K. Fu, "WattsUpDoc: Power side channels to nonintrusively discover untargeted malware on embedded medical devices," in *2013 USENIX Workshop on Health Information Technologies*, Aug. 2013.

[14] L. F. Combita, A. A. Cardenas, and N. Quijano, "Mitigating sensor attacks against industrial control systems," *IEEE Access*, vol. 7, pp. 92 444–92 455, 2019.

[15] D. Conroy, *Ransomware: Protecting your business from increasing cyberthreats*, 2024. [Online]. Available: `https://www.nar.realtor/blogs/emerging-technology/ransomware-protecting-your-business-from-increasing-cyberthreats`.

[16] D. J. Cook, *Activity Learning from Sensor Data*. Wiley, 2015.

[17] R. Dathathri, O. Saarikivi, H. Chen, K. Laine, K. Lauter, S. Maleki, M. Musuvathi, and T. Mytkowicz, "CHET: An optimizing compiler for fully-homomorphic neural-network inferencing," ser. PLDI 2019, Phoenix, AZ, USA: Association for Computing Machinery, 2019, ISBN: 9781450367127. DOI: `10.1145/3314221.3314628`. [Online]. Available: `https://doi.org/10.1145/3314221.3314628`.

[18] D. Demmler, T. Schneider, and M. Zohner, "ABY - a framework for efficient mixed-protocol secure two-party computation," in *Proc. of Network and Distributed System Security Symposium*, Jan. 2015.

[19] F. Ding, H. Li, F. Luo, H. Hu, L. Cheng, H. Xiao, and R. Ge, "DeepPower: Non-intrusive and deep learning-based detection of IoT malware using power side channels," in *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*, 2020, pp. 33–46.

[20] R. Doshi, N. Apthorpe, and N. Feamster, "Machine learning DDoS detection for consumer internet of things devices," in *2018 IEEE Security and Privacy Workshops (SPW)*, 2018, pp. 29–35.

[21] N. Dowlin, R. Gilad-Bachrach, K. Laine, K. E. Lauter, M. Naehrig, and J. R. Wernsing, "CryptoNets: Applying neural networks to encrypted data with high throughput and accuracy," in *Proc. of the 33rd Int'l Conf. on Machine Learning*, 2016, pp. 201–210.

[22] T. Elgamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Transactions on Information Theory*, vol. 31, no. 4, pp. 469–472, 1985. DOI: 10.1109/TIT.1985.1057074.

[23] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *Proc. of the 22nd ACM SIGSAC Conf. on Computer and Communications Security*, 2015, pp. 1322–1333, ISBN: 9781450338325.

[24] S. S. G, A. Darki, M. Faloutsos, N. Abu-Ghazaleh, and M. Sridharan, "IDAPro for IoT malware analysis?" In *12th USENIX Workshop on Cyber Security Experimentation and Test*, Aug. 2019.

[25] J. Geiping, H. Bauermeister, H. Dröge, and M. Moeller, "Inverting gradients - how easy is it to break privacy in federated learning?" *CoRR*, vol. abs/2003.14053, 2020.

[26] R. C. Geyer, T. Klein, and M. Nabi, "Differentially private federated learning: A client level perspective," *CoRR*, vol. abs/1712.07557, 2017. arXiv: 1712.07557. [Online]. Available: http://arxiv.org/abs/1712.07557.

[27] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof-systems," in *Proc. of the 7th Annual ACM Symp. on Theory of Computing*, 1985, pp. 291–304, ISBN: 0897911512. DOI: 10.1145/22145.22178.

[28] M. Goyal, I. Sahoo, and G. Geethakumari, "HTTP botnet detection in IoT devices using network traffic analysis," in *Int'l Conf. on Recent Advances in Energy-efficient Computing and Communication*, 2019, pp. 1–6.

[29] G. Gu, P. Porras, V. Yegneswaran, and M. Fong, "BotHunter: Detecting malware infection through IDS-Driven dialog correlation," in *16th USENIX Security Symposium (USENIX Se-*

curity 07), Boston, MA: USENIX Association, Aug. 2007. [Online]. Available: `https://www.usenix.org/conference/16th-usenix-security-symposium/bothunter-detecting-malware-infection-through-ids-driven`.

[30] S. Guynes, J. Parrish, and R. Vedder, "Edge computing societal privacy and security issues," *SIGCAS Comput. Soc.*, vol. 48, no. 3–4, pp. 11–12, Feb. 2020.

[31] I. Hafeez, M. Antikainen, A. Y. Ding, and S. Tarkoma, "IoT-KEEPER: Detecting malicious IoT network activity using online traffic analysis at the edge," *IEEE Trans. on Network and Service Management*, vol. 17, no. 1, pp. 45–59, 2020.

[32] L. Hanzlik, Y. Zhang, K. Grosse, A. Salem, M. Augustin, M. Backes, and M. Fritz, "MLCapsule: Guarded offline deployment of machine learning as a service," *CoRR*, vol. abs/1808.00590, 2018. arXiv: `1808.00590`. [Online]. Available: `http://arxiv.org/abs/1808.00590`.

[33] H. Ismail 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, 2019.

[34] C. Jindal, C. Salls, H. Aghakhani, K. Long, C. Kruegel, and G. Vigna, "Neurlux: Dynamic malware analysis without feature engineering," in *Proc. of the 35th Annual Computer Security Applications Conf.*, 2019, pp. 444–455.

[35] W. Jung, Y. Feng, S. A. Khan, C. Xin, D. Zhao, and G. Zhou, "DeepAuditor: Distributed online intrusion detection system for IoT devices via power side-channel auditing," *CoRR*, vol. abs/2106.12753, 2021. [Online]. Available: `https://arxiv.org/abs/2106.12753`.

[36] W. Jung, H. Zhao, M. Sun, and G. Zhou, "IoT botnet detection via power consumption modeling," *Smart Health*, vol. 15, 2020.

[37] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, “GAZELLE: A low latency framework for secure neural network inference,” in *Proc. of the 27th USENIX Conf. on Security Symposium*, 2018, pp. 1651–1668.

[38] L. Kang and H. Shen, “Attack detection and mitigation for sensor and can bus attacks in vehicle anti-lock braking systems,” in *29th Int’l Conf. on Computer Communications and Networks*, 2020, pp. 1–9.

[39] S. A. Khan, Z. Li, W. Jung, Y. Feng, D. Zhao, C. Xin, and G. Zhou, “DeepShield: Lightweight privacy-preserving inference for real-time IoT botnet detection,” in *37th IEEE International System-on-Chip Conference*, 2024.

[40] H. Kim, J. Smith, and K. G. Shin, “Detecting energy-greedy anomalies and mobile malware variants,” in *Proceedings of the 6th International Conference on Mobile Systems, Applications, and Services*, 2008, pp. 239–252.

[41] C. Kolias, G. Kambourakis, A. Stavrou, and J. Voas, “DDoS in the IoT: Mirai and other botnets,” *Computer*, vol. 50, no. 7, pp. 80–84, 2017.

[42] R. Lehmkuhl, P. Mishra, A. Srinivasan, and R. A. Popa, “MUSE: Secure inference resilient to malicious clients,” in *30th USENIX Security Symposium (USENIX Security 21)*, USENIX Association, Aug. 2021, pp. 2201–2218, ISBN: 978-1-939133-24-3.

[43] S. Li, K. Xue, C. Ding, X. Gao, D. S. L. Wei, T. Wan, and F. Wu, “FALCON: A fourier transform based approach for fast and secure convolutional neural network predictions,” *CoRR*, vol. abs/1811.08257, 2018. arXiv: `1811.08257`.

[44] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, “Federated learning: Challenges, methods, and future directions,” *CoRR*, vol. abs/1908.07873, 2019. arXiv: `1908.07873`. [Online]. Available: `http://arxiv.org/abs/1908.07873`.

[45] Z. Li, B. Perez, S. A. Khan, B. Feldhaus, and D. Zhao, “A new design of smart plug for real-time IoT malware detection,” in *2021 IEEE Microelectronics Design & Test Symposium (MDTS)*, 2021, pp. 1–6.

[46] Z. Li and D. Zhao, “ThingNet: A lightweight real-time mirai IoT variants hunter through cpu power fingerprinting,” in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2022.

[47] J. Lin, L. Zhu, W.-M. Chen, W.-C. Wang, C. Gan, and S. Han, “On-device training under 256kb memory,” *Advances in Neural Information Processing Systems*, vol. 35, pp. 22 941–22 954, 2022.

[48] J. Liu, M. Juuti, Y. Lu, and N. Asokan, “Oblivious neural network predictions via MiniONN transformations,” in *Proc. of ACM SIGSAC Conf. on Computer and Communications Security*, 2017, pp. 619–631.

[49] Y. Liu, L. Wei, Z. Zhou, K. Zhang, W. Xu, and Q. Xu, “On code execution tracking via power side-channel,” in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 1019–1031.

[50] Y. Liu, S. Ma, Y. Aafer, W.-C. Lee, J. Zhai, W. Wang, and X. Zhang, “Trojaning attack on neural networks,” in *In 25th Annual Network and Distributed System Security Symp.*, 2018.

[51] G. D. Maayan., *The IoT rundown for 2020: Stats, risks, and solutions*, https://securitytoday.com/articles/2020/01/13/the-iot-rundown-for-2020.aspx, [Online; posted January-13-2020].

[52] C. D. McDermott, F. Majdani, and A. V. Petrovski, “Botnet detection in the internet of things using deep learning approaches,” in *2018 Int’l Joint Conf. on Neural Networks*, 2018, pp. 1–8.

[53] H. B. McMahan, E. Moore, D. Ramage, and B. A. y Arcas, “Federated learning of deep networks using model averaging,” *CoRR*, vol. abs/1602.05629, 2016. arXiv: 1602.05629. [Online]. Available: http://arxiv.org/abs/1602.05629.

[54] Y. Meidan, M. Bohadana, Y. Mathov, Y. Mirsky, A. Shabtai, D. Breitenbacher, and Y. Elovici, “N-BaIoT—network-based detection of IoT botnet attacks using deep autoencoders,” *IEEE Pervasive Computing*, vol. 17, no. 3, pp. 12–22, 2018.

[55] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, “Kitsune: An ensemble of autoencoders for online network intrusion detection,” in *Proc. of Network and Distributed System Security Symp.*, Jan. 2018.

[56] P. Mohassel and Y. Zhang, “SecureML: A system for scalable privacy-preserving machine learning,” in *2017 IEEE Symposium on Security and Privacy (SP)*, Los Alamitos, CA, USA: IEEE Computer Society, May 2017, pp. 19–38.

[57] T. D. Nguyen, S. Marchal, M. Miettinen, M. H. Dang, N. Asokan, and A. Sadeghi, “DÏoT: A federated self-learning anomaly detection system for IoT,” in *IEEE 39th Int’l Conf. on Distributed Computing Systems*, 2019, pp. 756–767.

[58] R. Oak, M. Du, D. Yan, H. Takawale, and I. Amit, "Malware detection on highly imbalanced data through sequence modeling," in *Proceedings of the 12th ACM Workshop on artificial intelligence and security*, 2019, pp. 37–48.

[59] G. Oded, *Foundations of Cryptography: Volume 2, Basic Applications*, 1st. Cambridge Univ. Press, 2009, ISBN: 052111991X.

[60] S. I. Popoola, R. Ande, B. Adebisi, G. Gui, M. Hammoudeh, and O. Jogunola, "Federated deep learning for zero-day botnet attack detection in IoT-edge devices," *IEEE Internet of Things Journal*, vol. 9, no. 5, pp. 3930–3944, 2022. DOI: `10.1109/JIOT.2021.3100755`.

[61] V. Rey, P. M. S. Sánchez, A. H. Celdrán, G. Bovet, and M. Jaggi, "Federated learning for malware detection in IoT devices," *CoRR*, vol. abs/2104.09994, 2021.

[62] M. S. Riazi, M. Samragh, H. Chen, K. Laine, K. E. Lauter, and F. Koushanfar, "XONN: xnor-based oblivious deep neural network inference," *CoRR*, vol. abs/1902.07342, 2019. arXiv: `1902.07342`. [Online]. Available: `http://arxiv.org/abs/1902.07342`.

[63] M. S. Riazi, C. Weinert, O. Tkachenko, E. M. Songhori, T. Schneider, and F. Koushanfar, "Chameleon: A hybrid secure computation framework for machine learning applications," in *Proc. of Asia Conf. on Computer and Communications Security*, 2018, pp. 707–721.

[64] K. Rieck, P. Trinius, C. Willems, and T. Holz, "Automatic analysis of malware behavior using machine learning," *J. Comput. Secur.*, vol. 19, no. 4, pp. 639–668, Dec. 2011.

[65] B. D. Rouhani, M. S. Riazi, and F. Koushanfar, "DeepSecure: Scalable provably-secure deep learning," *CoRR*, vol. abs/1705.08963, 2017.

[66] G. Sagirlar, B. Carminati, and E. Ferrari, "AutoBotCatcher: Blockchain-based p2p botnet detection for the internet of things," in *IEEE 4th Int'l Conf. on Collaboration and Internet Computing (CIC)*, 2018, pp. 1–8.

[67] B. Schneier, *Applied Cryptography: Protocols, Algorithms and Source Code in C*, 20th. John Wiley, 2015.

[68] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," *CoRR*, vol. abs/1610.05820, 2016.

[69] L. Singh and M. Hofmann, "Dynamic behavior analysis of android applications for malware detection," in *2017 Int'l Conf. on Intelligent Communication and Computational Techniques*, 2017, pp. 1–7.

[70] S. Sriram, R. Vinayakumar, M. Alazab, and S. KP, "Network flow based IoT botnet attack detection using deep learning," in *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications Workshops*, 2020, pp. 189–194.

[71] D. Strom, *How the mirai botnet continues to threaten business networks*, `https://https://siliconangle.com/2023/05/30/mirai-botnet-continues-threaten-business-networks`, [Online; posted May-30-2023].

[72] J. Su, D. V. Vasconcellos, S. Prasad, D. Sgandurra, Y. Feng, and K. Sakurai, "Lightweight classification of IoT malware based on image recognition," in *IEEE 42nd Annual Computer Software and Applications Conf.*, vol. 02, 2018, pp. 664–669.

[73] H. Sun, X. Wang, R. Buyya, and J. Su, "CloudEyes: Cloud-based malware detection with reversible sketch for resource-constrained internet of things IoT devices," *Software: Practice and Experience*, vol. 47, pp. 421–441, 2017. DOI: `10.1002/spe.2420`.

[74] Texas Instruments, *Bidirectional current/power monitor with I^2C interface*. [Online]. Available: https://www.ti.com/product/INA219/.

[75] S. Tople, K. Grover, S. Shinde, R. Bhagwan, and R. Ramjee, "Privado: Practical and secure DNN inference," *CoRR*, vol. abs/1810.00602, 2018. arXiv: 1810.00602. [Online]. Available: http://arxiv.org/abs/1810.00602.

[76] F. Tramèr and D. Boneh, *Slalom: Fast, verifiable and private execution of neural networks in trusted hardware*, 2019. arXiv: 1806.03287 [stat.ML].

[77] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Stealing machine learning models via prediction apis," *CoRR*, vol. abs/1609.02943, 2016.

[78] S. Wagh, D. Gupta, and N. Chandran, "SecureNN: 3-party secure computation for neural network training," *Proceedings on Privacy Enhancing Technologies*, vol. 2019, pp. 26–49, Jul. 2019. DOI: 10.2478/popets-2019-0035.

[79] X. Wei, Y. Liang, X. Li, C. H. Yu, P. Zhang, and J. Cong, "TGPA: Tile-grained pipeline architecture for low latency cnn inference," in *IEEE/ACM Int'l Conf. on Computer-Aided Design*, 2018, pp. 1–8.

[80] R. Xu, N. Baracaldo, Y. Zhou, A. Anwar, S. Kadhe, and H. Ludwig, *DeTrust-FL: Privacy-preserving federated learning in decentralized trust setting*, 2022. arXiv: 2207.07779 [cs.CR].

[81] H. Yang, "H-FL: A hierarchical communication-efficient and privacy-protected architecture for federated learning," *CoRR*, vol. abs/2106.00275, 2021. arXiv: 2106.00275. [Online]. Available: https://arxiv.org/abs/2106.00275.

[82] Q. Yang, P. Gasti, G. Zhou, A. Farajidavar, and K. S. Balagani, “On inferring browsing activity on smartphones via usb power analysis side-channel,” *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 5, pp. 1056–1066, 2017.

[83] A. C.-C. Yao, “How to generate and exchange secrets,” in *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, 1986, pp. 162–167. DOI: `10.1109/SFCS.1986.25`.

[84] J. Zhang, B. Chen, Y. Zhao, X. Cheng, and F. Hu, “Data security and privacy-preserving in edge computing paradigm: Survey and open issues,” *IEEE Access*, vol. 6, pp. 18 209–18 237, 2018.

[85] Q. Zhang, C. Wang, H. Wu, C. Xin, and T. Phuong, “GELU-Net: A globally encrypted, locally unencrypted deep neural network for privacy-preserved learning,” in *Proc. of Int’l Joint Conf. on Artificial Intelligence*, Jul. 2018, pp. 3933–3939.

[86] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, “Federated learning with non-IID data,” *CoRR*, vol. abs/1806.00582, 2018. arXiv: `1806.00582`. [Online]. Available: `http://arxiv.org/abs/1806.00582`.

[87] L. Zhu, Z. Liu, and S. Han, “Deep leakage from gradients,” *CoRR*, vol. abs/1906.08935, 2019.

VITA

Sabbir Ahmed Khan

Department of Computer Science

Old Dominion University

Norfolk, VA 23529

PATENT

1. G. Zhou, W. Jung, C. Xin, D. Zhao, Y. Feng, and S. A. Khan, "Privacy-preserving online botnet classification system utilizing power footprint of IoT connected devices," Patent office: US, Patent number: 12015622, June 18, 2024.

PUBLICATIONS

1. S. A. Khan, Z. Li, W. Jung, Y. Feng, D. Zhao, C. Xin, and G. Zhou, "DeepShield: Lightweight Privacy-Preserving Inference for Real-Time IoT Botnet Detection" in 37th IEEE International System-on-Chip Conference (SOCC), September 16-19, 2024. (Accepted)
2. W. Jung, Y. Feng, S. A. Khan, C. Xin, D. Zhao and G. Zhou, "DeepAuditor: Distributed Online Intrusion Detection System for IoT Devices via Power Side-channel Auditing," in 2022 21st ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN), Milano, Italy, 2022, pp. 415-427, doi: 10.1109/IPSN54338.2022.00040.
3. Google scholar link: https://scholar.google.com/citations?user=mJNY8gIAAAAJ&hl=en, to see the full list of publications