July 2021

Security Attacks and Defenses in Cyber Systems: From an AI Perspective

Zhengping Luo
University of South Florida

Security Attacks and Defenses in Cyber Systems: From an AI Perspective

by

Zhengping Luo

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
Department of Electrical Engineering
College of Engineering
University of South Florida

Major Professor: Zhuo Lu, Ph.D.
Mahshid Rahnamay Naeini, Ph.D.
Jay Ligatti, Ph.D.
Kwang-Cheng Chen, Ph.D.
Kaiqi Xiong, Ph.D.

Date of Approval:
June 29, 2021

Keywords: Cybersecurity, Attacks and Defenses, Adversarial Machine Learning, Cognitive Radio Networks, High Performance Computing, Binary Code Similarity Detection

Dedication

This dissertation is dedicated to my dear family, including my parents, my two brothers and two sisters-in-law, and especially to my nephew, Haoran Luo(罗浩然), who was born when I was in my early PhD years, and also, to my on-the-way nephews and nieces. If without the unconditional mental and economic support of my family all the way, I could not have any chance to make it here.

I want to give my sincere thanks to some of my so many great teachers I have met at different stages of my student journey: Teacher Yihua Luo(罗益华老师), Teacher Changzheng Zeng(曾长征老师), Teacher Xiling Liu(刘希岭老师) and Professor Tianqi Yang(杨天奇老师). I could not imagine what my life will be if without their tremendous guidance and their faith in me in my academic journey.

During the past four and a half years, I have received so much support from my long-time friends, especially Liman Zeng(曾立满), Hongfei Wang(汪洪飞), Yan Li(李炎), Yanjun Liu(刘延钧). I met them at different time of my life. If without so many chats and helps from them, my PhD life would be much boring and bitter. Part of the thanks is also going to so many great friends I have met during my PhD years, especially Tianze Guo (郭天泽) and Fa Zheng(郑发).

Acknowledgments

PhD life is quite challenging and tough, but for me, it is also such a memorable journey full of joy and gratitude. My supervisor, Dr. Zhuo Lu absolutely has much of my appreciation. His down-to-earth working style shaped me a lot in doing research. My so many meetings with him in his office are full of short, straightforward, intuitive, constructive and insightful suggestions from him. During my whole PhD years, I never worried about funding issues, and I seldom worried about my career, as I know I have the best supervisor and team I could ever have with me.

Dr. Shangqing Zhao, my academic brother, during these years, he taught me how to drive hand-by-hand, he showed to me how to do research from a practical perspective, he shared with me tons of his insightful understandings about research, about life and we had so many enjoyable conversations.

It is my single greatest honor and fortune to have Dr. Lu as my supervisor and Dr. Zhao as my academic brother during my PhD years.

Special appreciation is given to my PhD committee members: Dr. Kwang-Cheng Chen, Dr. Jay Ligatti, Dr. Mahshid Rahnamay Naeini and Dr. Kaiqi Xiong. I feel so humble and honored to have them as my committee members and for their suggestions.

I want to give my unique thanks to my colleagues and also my friends I met during these years, including Tao Hou, Rui Duan, Zhe Qu, Mohammed Alrowaily, Mingchen Li, Keyu Chen, Hung Nguyen and Dumindu Samaraweera etc., during my PhD years.

Table of Contents

List of Tables

List of Figures

Abstract

Security of real-world cyber systems has drawn a lot of attention in recent years, especially when machine learning techniques are widely deployed into different layers of cyber systems. With the technology of machine learning, especially adversarial machine learning techniques, the attacks and defenses in cyber systems have shown a lot of new characteristics. In this dissertation, two major works regarding the attacks and defenses in real world cyber systems including dynamic spectrum sensing systems and High Performance Computing (HPC) systems and software systems are discussed.

In the first work, we revisit this security vulnerability of cooperative spectrum sensing as an adversarial machine learning problem and propose a novel learning-empowered framework named Learning-Evaluation-Beating (LEB) to mislead fusion centers. Given the gap between the new LEB attack and existing defenses, we introduced a non-invasive and parallel method named influence-limiting defense sided with existing defenses to defend against LEB-based or other similar attacks.

In the second work, we offer a novel perspective, treating the anomaly detection in HPC systems based on log files as a sequential decision process, and further applying reinforcement learning techniques to detect anomalies or malicious users. Start from there, we also provide a binary code similarity detection-based method that can be applied to a more general scenario in software systems through utilizing Recurrent Neural Network (RNN) and Siamese Neural Network to detect malwares from the binaries generated by the processor that executing the program.

Chapter 1: Introduction

As cyber security becomes a more and more widely discussed topic in our daily life, the attacks and defenses of real-world cyber systems come out as a real challenge for both the industries and academia. Especially with the widely adoption of machine learning techniques, new challenges are brought to the safely deployment of cyber infrastructures. In this dissertation, we explored the attacks and defenses of cyber systems from a machine learning perspective such as dynamic spectrum sensing systems, High Performance Computing (HPC) systems and software systems.

1.1 Security Attacks and Defenses in Dynamic Spectrum Sensing Systems

Cooperative spectrum sensing has been proposed as an effective mechanism to enhance the spectrum sensing performance using cognitive radio devices (e.g., TV-band devices coded in IEEE 802.22). It enables a data fusion-based decision framework, in which multiple nodes report their sensing results to a fusion center that makes a centralized decision to enhance the spectrum sensing accuracy. This, however, opens up opportunities for Byzantine attacks (also widely referred to as spectrum sensing data falsification attacks) [9, 10, 11, 12, 13, 14, 15, 16], in which attackers aim to send malicious sensing results to mislead the fusion center to make wrong decisions.

Defense mechanisms to combat Byzantine attacks have been widely studied and include: (i) statistics-based mechanisms, i.e., building statistical models to detect or eliminate attacks [12, 17, 18, 19]; (ii) machine learning-based mechanisms, i.e., using machine learning methods as countermeasures[20, 14, 21, 10, 22]; (iii) trust (or reputation)-based mechanisms, i.e.,

building trust metrics for nodes to be weighted in the decision process to identify or mitigate attacks [11, 13].

Nevertheless, existing studies mainly make network or attack assumptions that often biased towards defenses. For example, methods used in [12, 17, 19] assume that attacks behave in a particular way or prior information of attack statistics is known such that a statistical model of an attack can be built; and machine learning based methods [14, 21, 10, 22] assume that malicious data pattern deviates from normal data pattern under a given classification rule. These assumptions give an advantage to the defenses over the attacks. However, in practice, attackers can try to actively avoid pre-assumed behaviors or break assumptions used in the defense. Moreover, the time-varying nature of wireless channels and signals can deviate the data properties of the legitimate sensing results from time to time. In this regard, training and prior statistical models used by the defenses face a model mismatch phenomenon over time, which can also be exploited by attackers. All these motivate us to rethink Byzantine attacks from a new perspective: is there a stronger attack model?

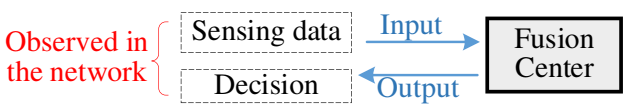


Figure 1.1. Abstract model of data processing in cooperative spectrum sensing.

Our new observations on cooperative spectrum sensing are twofold: (i) Sensing nodes always report their sensing data as the input to the fusion center; and (ii) regardless of what kinds of defense mechanisms it adopts, the fusion center always announces the final decision, which is the output from the fusion center. If we treat the fusion center as a black box, the defense and decision rules together in the fusion center can be considered as a black-box function with known inputs and outputs, as illustrated in Figure. 1.1. Inspired by the *no free lunch* theorem [23] and the *transferability* [24] property in machine learning, attackers can, in fact, use the inputs and outputs shown in Figure. 1.1 to build a surrogate model of the

targeted fusion center, and then launch effective attacks with minimum data manipulation to mislead the fusion center. However, the model is essentially a partial model of the fusion center.

To make the partial model to attack cooperative spectrum sensing process, we propose a *Learn-Evaluate-Beat (LEB)* framework in this work. As its name indicates, LEB consists of three steps: (i) the initial learning step, in which the attacker uses incremental learning models to build its own surrogate model to approximate the fusion center's decision model; (ii) the evaluation step, in which the attacker evaluates whether its own model is accurate enough or not to launch attacks; and (iii) the final beating step, in which the attacker falsifies the sensing data with the minimum cost to change the fusion center's decision if they pass the evaluation step. To design LEB attacks to be effective, we also propose a learning algorithm based on a set of sub-models and a generic data generation algorithm to generate adversarial examples (i.e., falsified data with the minimum data manipulation to flip the fusion center's decision) in sub-models. We conduct comprehensive real-world experiments to measure the performance of spectrum sensing under LEB attacks and their impacts against a wide range of existing defense methods.

Towards the defense perspective, an effective defense mechanism to distinguish LEB-based or similar malicious nodes out from normal nodes is quite challenging. The principal reason that LEB attackers can succeed contributes to that the malicious nodes can build up their influence or impact on the fusion center through taking advantage of the model mismatch phenomenon. We introduce a new metric named as decision flipping influence to measure the influence of a given subset of sensing nodes. Based on the decision flipping influence, we further propose an influence-limiting policy to evaluate and limit the influence any subset of nodes can have on the fusion center, thus decreasing or eliminating the attack capability of the malicious nodes. The influence-limiting policy is a non-invasive, parallel method sided with traditional/existing defenses. Our experimental results demonstrate that

the proposed defense can effectively bridge the gap between traditional defenses in cooperative spectrum sensing and the new LEB-based attacks.

Given the shortcomings of enforcing the influence-limiting policy in real-world, we offer a solution to decrease the computational complexity of influence-limiting defense. We show that this low-cost version of influence-limiting defense can reduce the computational significantly. On the average, the low-cost version needs less than 20% of the time required by the original influence-limiting defense while still maintaining roughly the same level of defense performance as the original defense.

1.2 Security in High Performance Computing (HPC) and Software Systems

High performance computing, a widely used technology in fields such as aerospace, automotive, semiconductor design, energy explore, financial computing, weather forecast and nuclear simulation etc., mainly focused on how to improve the performance of the computing from software development, parallel algorithms and computer architecture etc. It has become an important way to do scientific research. Clusters and grids are the two dominant and distinct methods of deploying HPC parallelism in both industrial and academia nowadays [25].

As performances and capabilities of HPC infrastructures have continued to evolve, applications and software that are running on HPC infrastructures have also increased, especially with software applications of open-source frameworks. However, they also become increasingly desirable targets to attackers [26]. HPC security, which is one of the most concerned problems for both users and HPC system maintainers, is not a significant problem in the past. Because most of HPC systems were built for private use by some dedicated users at the beginning. Now it has been the main worries and obstacles for the spreading of HPC systems as now they are often shared by multiple users, especially with the emerging of the idea of HPC-as-a-service.

The security of HPC systems has long been a research focus to build reliable and accountable systems [27, 28, 29, 30, 31], especially as log files being introduced to detect malicious intruders and behaviors. Computing power, as a resource, is required to be used only by those approved users, or further, to prevent approved users from performing unapproved behaviors. On the other hand, legal users of computing resources want to protect communications between users and resource owners to be safe and keep their data away from malicious attackers due to most of jobs and resources run or stored in HPC systems usually are sensitive and high-profit information. It might have serious consequences when hacked. Other security problems like denial-of-service are also a concern for HPC systems that has a lot of access requests from users.

Typically, HPC systems can be decoupled into 4 layers [32, 33]: application layer, middleware layer, operating system layer and network layer. In nowadays systems, each of the 4 layers has been well-instrumented for security evaluation. Information like percentage of CPU time, memory utilization, I/O time, MPI operations, access information etc. are all well-logged and can be used to analyze and evaluate the performance and security of HPC. Log analytics in HPC has been a more straightforward and effective strategy to secure HPC systems against malicious users [4, 31, 34].

There are many anomaly/intrusion detection methods based on log analytics in HPC have been proposed [31, 3, 35], especially when machine learning techniques are involved [4, 34]. Machine learning techniques such as deep learning, support vector machine (SVM) have their unique advantages in dealing with large volumes of logs generated by HPC on each computing node [5].

Nevertheless, anomaly/intrusion detection based on log analytics in HPC is not quite the same as traditional classification problems. As the logs in HPC are generated continuously when the system is running, thus the input of anomaly/intrusion detection mechanism is also streaming data. We need to parse the streaming data into tokens, which is defined as a

small group of log lines, based on which we can extract feature vectors. Further the feature vectors can be used to perform anomaly/intrusion detection.

The decision of whether the evaluated user is malicious or not depends not only the pattern of all feature vectors, but also the order. Therefore, we can treat detection process of anomaly/intrusion detection in HPC as a sequential decision process, in which final decision of the normality of a user depends on a set of small decisions based on all previous feature vectors. When we think the problem from a sequential decision perspective, we can employ reinforcement learning techniques to model the anomaly/intrusion detection problem in HPC.

In this work, we propose a framework, named as ReLog, based on reinforcement learning techniques to perform log analytics in HPC for anomalous user detection. We observed that MPI operations used by computing nodes are usually a subset of pre-defined operations [8]. Thus we construct feature vectors based on MPI logs and treat them as different states in reinforcement learning. Feature vectors can be obtained from parsing of logs using sliding windows. The ultimate reward comes from final classification decision. When malicious users are detected, the reward will reach a maximum value.

From a more general perspective, identification of bugs or other security-related vulnerabilities in software projects still remains to be a challenging problem in software security. Even though intensive work has been conducted by both the research and industry communities[36, 37, 38], the security of software projects still poses great concerns given that many software projects are used by millions or even billions of users. A small bug, such as the failure of checking buffer boundaries could lead to disastrous consequences [39, 40].

These kinds of vulnerabilities in software projects could be resulted from mistakes of trusted program developers, or from malicious attackers. Thanks to the intellectual property or security concerns, many of those software providers will not disclose their source code to

the public. Therefore, from the users' perspective, analyzing the binary code when executing the software becomes their practical option to evaluate the security of software projects.

Binary code similarity detection could be applied to different scenarios, while malware detection is one of them, other applications include plagiarism detection[37] and vulnerability detection [41]. Extensive efforts [2, 42, 36] have been given to detect similar functions from binaries that generated from different platforms, such as x86, ARM and MIPS.

The general working flow of binary code similarity detection [2, 43, 42] focuses on extracting representative local feature vectors for each node in the control flow graphs (CFGs) that disassembled from binary code. Various statistical features and block features are proposed to represent the graph blocks [2, 43]. Then these local feature vectors are encoded into embeddings such that further similarity comparison techniques could be applied to perform the detection or comparison. One important type of the similarity comparison method is the graph-based matching algorithms. For example, using bipartite matching algorithm to calculate the similarity of two CFGs [2].

However, graph matching-based approaches have two shortcomings according to [1]: (i) It is less flexible for the similarity functions approximated through graph matching techniques to adapt to other applications; (ii) The graph matching algorithms suffer from low efficiency, thus leads to the inefficiency of the similarity detection process. These two shortcomings make the graph matching-based method challenging to be applied in a broader perspective. There are also some deep neural network-based methods have been proposed to embed the local features [36, 41, 44, 1]. However, most of them involves training a deep graph neural network or sematic-aware neural network, which usually complicates the problem especially in terms of computation cost.

In binary code similarity detection, the first step is to extract local features and encode them into embeddings. Given the shortcomings of graph matching-based approaches and deep neural network-based methods, in this work, we propose a new embedding method

based on long short-term memory (LSTM) recurrent neural network (RNN), which is widely applied to sequence classification [45] and pattern-based feature selection/classification [46].

The sequence of local block features can be fed into an LSTM recurrent neural network. Through multiple rounds of training, the final cell state of the LSTM recurrent neural network will learn much of the information from all the previous trained input and output pairs, which could be utilized as the final embeddings for the binaries and be further analyzed using classification techniques. As we found in our experimental results that LSTM network-based methods often could achieve similar performance with less training and computation cost.

For the similarity comparison, the traditional methods including Euclidean distance comparison and classification methods focus on the statistical characteristics of two inputs. However, the embedding we extracted from graph-based embedding methods or neural network-based methods often contain much information that traditional methods fall short of capturing. To overcome this weakness and fully capture the information, we employ Siamese neural network [47] to perform the similarity measurement. The main advantage of the Siamese neural network is it employs a unique structure to learn the embedding such that the semantic similarity could be learned thus placing the same classes of the input close together.

1.3 Contributions of the Thesis

Our contributions are listed as follows.

- We rethink the traditional security of cooperative spectrum sensing and present a new perspective to create a stronger attack model named as LEB attack. Our work has shown that the traditional duel of attacks and defenses in cooperative spectrum sensing has to be re-visited in the presence of new learning-based attack models.

- For the LEB framework to be effective and practical, the framework is designed in a flexible way to adopt a wide range of sub-models to build the surrogate model of the fusion center. A generic generation algorithm is proposed to create adversarial examples against cooperative spectrum sensing.

- Our experiments show that the LEB attack can achieve an up to 82% attack success ratio while only manipulating a small number of malicious nodes. The proposed adversarial data generation algorithm achieves similar performances as existing generation methods while reducing by 65% of the computational cost on average.

- We introduce a generic non-invasive method, the influence-limiting policy, sided with traditional/existing defenses to counter LEB attacks or other similar learning-based attacks. Experimental results of influence-limiting policy-based defense demonstrate an average overall disruption ratio reduction by 78% under LEB attacks compared with traditional defenses while without the influence-limiting policy.

- We conduct a comprehensive review on the security of HPC systems, including vulnerabilities, consequences, and potential solution strategies. Well-known attacks and defense methods, especially intrusion/anomaly detection methods, in HPC systems are identified.

- We build a reinforcement learning-based framework named as ReLog to perform log analytics in HPC. We provide a new perspective, treating the anomaly/intrusion detection as a sequential decision problem, to detect anomalous/malicious users.

- Based on GAN, we give a solution to generating sufficient training data based on available logs, such that deep neural network models can be fully trained.

- We collect real-world MPI logs and perform comprehensive experiments to validate ReLog and compared it with existing other anomaly/intrusion detection mechanisms.

The results show that ReLog can achieve 93% of the detection accuracy on our collected dataset.

- We proposed an LSTM recurrent neural network-based model to embed local feature vectors of CFGs disassembled from binaries, which captures the general information of all the historical blocks and the dependence information, also makes the following similarity detection process more efficient.

- We employed Siamese neural network to perform the similarity measurement in binary code similarity detection, which learns the embedding semantic information and demonstrate multiple advantages compared with existing traditional similarity detection methods such as Euclidean distance.

- Comprehensive experiments are conducted based on real-world collected binary dataset and the experimental results validated our mechanism has a slightly better performance with existing methods such as graph matching-based methods in terms of detection accuracy and less computation cost compared with existing deep neural network-based methods.

1.4 Dissertation Overview

In Chapter 2, we presented the security attacks and defenses in dynamic spectrum sensing systems, and we offered the proposed LEB attack framework based on adversarial machine learning techniques, and we also gave the proposed defense strategy, influence-limiting defense, to counter the learning-empowered attack. In Chapter 3, we offered the anomaly detection method based on the log files of HPC systems and build a reinforcement learning based framework to detect malwares. Also, we elaborated the binary code similarity detection method based on LSTM and Siamese Neural Networks in this chapter. In Chapter 4, we conclude the dissertation.

Chapter 2: Attacks and Defenses in Dynamic Spectrum Sensing

In this chapter, we explored the security attacks and defenses in dynamic spectrum sensing systems. This chapter was published in IEEE Transactions on Mobile Computing and the ACM workshop on Wireless Security and Machine Learning (WiseML 21). Permissions are included in Apendix A.

2.1 Model and Preliminaries

In this section, we introduce the system model, overview existing studies, and identify challenges.

2.1.1 System Model

We consider a cooperative spectrum sensing scenario with n sensing nodes and one fusion center, as shown in Figure. 2.1. At each timeslot (i.e., each round of sensing), all nodes perform spectrum sensing on a TV spectrum channel, then report their results to the fusion center that makes the global channel usability decision based on all inputs. We assume that energy detection [17, 12] is employed at each node. A sensing report contains the value of the energy level sensed by the corresponding node.

The energy level vector at the ith timeslot from n sensing nodes is denoted by $\mathbf{x}_i = [x_{i,1}, x_{i,2}, ..., x_{i,n}]^\top, \mathbf{x}_i \in \mathcal{X} \subset \mathbb{R}^{n\times 1}$, where $[\cdot]^\top$ is the matrix transpose operator. $\mathcal{X} \subset \mathbb{R}^{n\times 1}$ is the value space for the sensed energy level, and $\mathbb{R}^{n\times 1}$ is an n-dimensional Euclidean space. Historical sensing results of all sensing nodes are denoted as $\{\mathbf{h}_k\}_{k\in[1,n]} = [x_{1,k}, x_{2,k}, x_{3,k}, ..., x_{i,k}]^\top$.

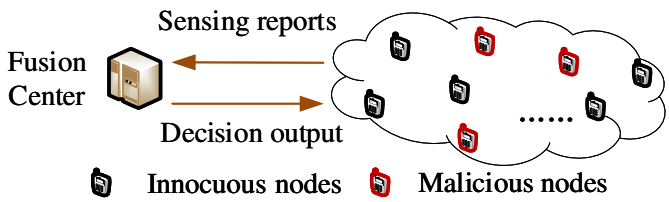


Figure 2.1. System model.

We denote the decision mapping function (implemented by the data fusion rule and potential defenses) used at the fusion center as $O : \mathcal{X} \rightarrow \mathcal{Y}$, where $\mathcal{Y} = \{-1, 1\}$ is the decision output space with $-1/1$ denoting the channel being available/unavailable. The fusion center makes the final channel status decision $y_i \in \mathcal{Y}$ based on sensed results $\mathbf{x}_i$. We assume the attacker has no information about the decision mapping function adopted by the fusion center while it knows only the final decision y_i at each timeslot i.

Cooperative spectrum sensing enables multiple nodes to report their sensing results to a fusion center to enhance the spectrum sensing accuracy. In the meantime, Byzantine nodes can send malicious sensing results to the fusion center and mislead the fusion center to make wrong decisions about the channel status and further cause interference between primary users of the spectrum with secondary users.

Without loss of generality, we assume that the first m (out of n) nodes are malicious nodes that are fully controlled by the attacker. The attacker can make malicious nodes to report whatever sensing results needed but has no information about the reports of remaining $n - m$ innocuous nodes.

We denote the sensed data vector by malicious nodes at timeslot i as $\mathbf{a}_i = [x_{i,1}, x_{i,2}, ..., x_{i,m}]^\top$, $\mathbf{a}_i \in \mathcal{A} \subset \mathbb{R}^{m \times 1}$, where $\mathcal{A}$ is the report space of malicious nodes. Obviously, $\mathbf{a}_i$ is the first part of $\mathbf{x}_i$ that

$$\mathbf{x}_i = [\underbrace{x_{i,1}, x_{i,2}, ..., x_{i,m}}_{\mathbf{a}_i}, x_{i,m+1}, ..., x_{i,n}]^\top, \mathbf{x}_i \in \mathcal{X}. \tag{2.1}$$

The objective of the attacker is to manipulate sensing data vector $\mathbf{a}_i$ to mislead the fusion center to yield a wrong sensing decision y_i.

2.1.2 Defending against Byzantine Attacks

Defenses against Byzantine attacks in cooperative spectrum sensing can be essentially viewed as an anomaly detection problem, in which anomalous (malicious) nodes need to be distinguished from innocuous nodes. Many of defenses proposed in previous works correspond to a combination of two or multiple aspects of node characteristics, such as transmit pattern and statistical consistency. Here, we classify these defenses into three broad categories by the main node characteristic used by defenses:

- *Statistics-based defense,* which assumes that the attack has a particular behavior or prior information of network or attack statistics are known [12, 17, 19]. As the most widely used approach to detect malicious nodes, statistics-based schemes defend fusion centers against attacks based on the statistical measures derived from the sensing report history $\{\mathbf{h}_k\}_{k\in[1,n]}$, such as covariance and deviation [12, 17, 18].

- *Machine learning-based defense,* in which malicious nodes are assumed to have different underlying data patterns, for example, historical sensing results $\{\mathbf{h}_k\}_{k\in[1,m]}$ of malicious nodes and $\{\mathbf{h}_k\}_{k\in[m+1,n]}$ of innocous nodes can be classified into different categories by machine learning techniques. Both supervised [14] and unsupervised [21, 10, 22] methods have been proposed to identify or eliminate the effects malicious nodes.

- *Trust (or reputation)-based defense,* the essence of which is to compute a trust metric based on $\{\mathbf{h}_k\}_{k\in[1,n]}$. Those nodes with low trust metrics are detected as malicious nodes and get eliminated or less weighted from the decision process. It is worth noting that trust-based methods usually compute trust metrics based on mathematical deviations obtained from statistics-based mechanisms [13, 11].

These defense methods have been demonstrated to be effective and practical in countering attacks. However, their common assumption is that attackers are passive, i.e., they follow a fixed or pre-assumed attack strategy. New learning-empowered active attackers will have the opportunity to take advantage of these defenses.

2.1.3 Model Mismatch in Realistic Network Scenarios

Given assumptions made by existing defenses against Byzantine attacks, the question is whether there is any model mismatch phenomenon between the assumed and actual (real-world) scenarios, i.e., whether the statistical properties and other measures like pattern and reputation for each node will differ, or not. We explore the model mismatch phenomenon both in spatial and temporal dimensions.

Two datasets are used in our exploration. The first one is a public dataset collected from 5282 different locations in Atlanta metropolitan area over 13 TV white space channels [48]. We use the dataset to explore the model mismatch phenomenon in spatial dimension. The other one is collected from a time period of 100 hours over 22 channels, 5 different locations were sensed over a 20×20 km^2 urban region. We use USRP N210 [49] as the sensor. This dataset is used to evaluate the model mismatch phenomenon in temporal dimension.

The model mismatch phenomenon in spatial dimension is illustrated in Figure. 2.2(a) and Figure. 2.2(b). Figure. 2.2(a) shows the signal strengths for one TV channel (randomly chosen from 13 channels) over 5282 different locations. The signal strength varies from -45dBm to -100dBm. Fig. 2.2(b) shows the probability distribution in terms of signal strengths of 5 locations from our collected dataset. It is obvious that locations C and E have the lowest signal strength while location B has the highest signal strength for the given channel.

The model mismatch phenomenon in temporal dimension is shown in Figure. 2.2(c). The 100-hour dataset is divided into 5 sub-datasets, each of which includes 20 hours of the data. The probability distributions of the signal strengths for each sub-dataset (averaged over 22

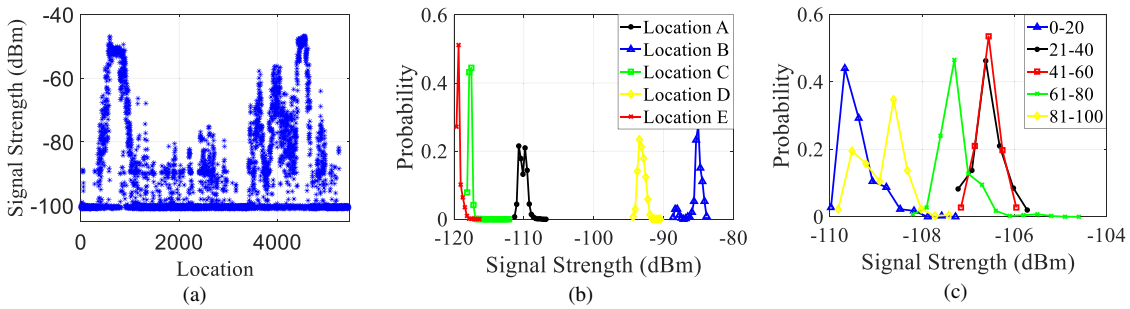


Figure 2.2. Signal strengths viration at different time and locations.

channels) are plotted in Figure. 2.2(c), which shows that the probability distribution changes over different sub-datasets. This observation indicates that a statistical or data model built for one time period changes for another time period.

Our data analysis on existing datasets and collected datasets demonstrate that the statistical/data model mismatch phenomenon over space and time is a real-world problem. In another words, it can be difficult for signal strengths of both malicious and innocuous nodes to follow exact models/behaviors assumed in a defense strategy. The existence of such phenomenon is also reasonable because of (i) environmental factors, such as weather and buildings, and (ii) network factors, such as co-channel interference and adjacent-channel leakages from other broadcasting activities.

The model mismatch phenomenon further leads to two consequences: (i) uncertainty of sensing results occur among multiple nodes (even among innocuous nodes) simultaneously, and (ii) it will be challenging to find values of parameters for the optimal decision to decide the channel status over time. Thus, an intelligent attacker can try to learn decision models, then takes advantage of the learned model and generates malicious sensing data to mislead the fusion center.

2.2 LEB Attack Framework: Motivation and Design

In this section, we first present the motivation behind the adversarial learning-based attack design and then propose the LEB attack framework against cooperative spectrum sensing.

2.2.1 Attack Motivation

As we can observe from the model mismatch phenomenon, it is often impractical for the fusion center to have perfect models about real-world signal data statistics or patterns due to the spatially and temporally varying nature of the wireless environment. Moreover, if we turn the table around and think from an intelligent attacker's perspective, the attacker should avoid any known behavior as assumed in traditional defenses and make its attack stealthy.

Our key observations for designing a new attack model are threefold: (i) network nodes always report their sensing data to the fusion center; (ii) as spectrum sensing applications are built upon the wireless scenario and the goal of sensing is to let nodes know the availability of a spectrum band (e.g., for them to access the channel opportunistically), it is reasonable to assume that the final decision is broadcast by the fusion center to all nodes to inform them of the decision to use (or not use) the band (iii) many of cooperative spectrum sensing methods [17, 12, 11, 10] are based on the fact that each node performs the channel sensing task independently and the fusion center tries to make a better decision given independent decisions from all nodes.

Based on the above three observations, we treat the fusion center as a black box such that the decision model (including both fusion rules and potential defenses) can be considered as a black-box function with known inputs and outputs. As a result, the attacker uses inputs and outputs of this back box to build a surrogate model of the decision model used at the

targeted fusion center. The third observation opens the door to use a small number of nodes ($m < n$) to build a surrogate model. If $m = n$, then the attacker has full control of the fusion center and the attack success ratio will be 100%. When $m < n$, it will be difficult for the attacker to achieve 100% of the attack success ratio, but there is still a probability to succeed. The design of the LEB attacker is based on this point. After stealing the decision model, the attacker can launch attacks with minimum data perturbation to mislead the fusion center with a high probability.

Our idea of the surrogate model is partially based on [50, 51, 52]. Provided that the learning capacity of the surrogate model is equivalent to or stronger than the decision model at the fusion center, it will be easier to gradually learn an approximation of the decision model and is also practically feasible as it does not require the prior knowledge of the decision model or training data.

2.2.2 LEB Attack Architecture

By leveraging the three observations in Section 2.2.1, we propose the *Learning-Evaluation-Beating (LEB)* attack framework based on adversarial machine learning to launch attacks against cooperative spectrum sensing. As the name indicates, the LEB attack consists of three steps: (i) *learning*, in which the attacker uses incremental learning models to build its own surrogate model to approximate the fusion center's decision model; (ii) *evaluation*, in which the attacker evaluates whether the selected surrogate model is accurate enough to launch attacks, and (iii) *beating*, in which adversarial sensing results are generated to fool the decision model. The flow chart of the LEB attack framework is shown in Figure. 2.3, in which the red-colored components are controlled by the attacker. It is worth noting that the attacker is defined as a malicious controller that can access all the reports of controlled and compromised nodes, and modify the information that each device reports to the fusion center.

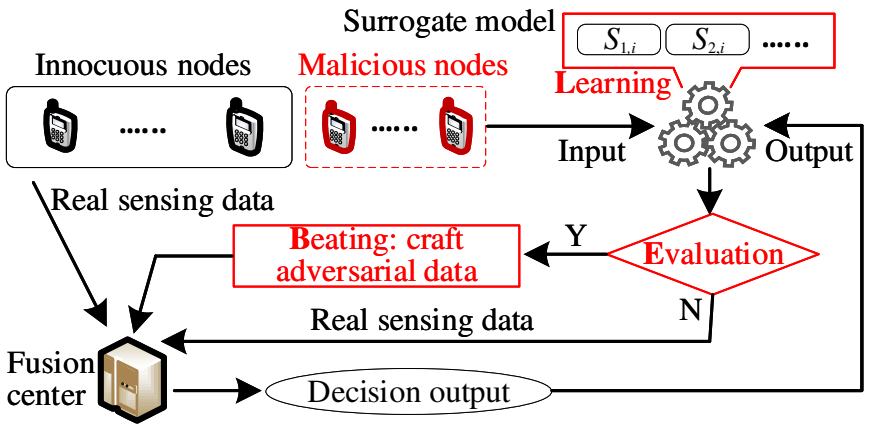


Figure 2.3. The LEB attack framework.

2.2.2.1 Learning

The learning step aims to build the surrogate model S_i at timeslot i to approximate the decision model O at the fusion center given malicious nodes' reporting vector $\{\mathbf{a}_j\}_{j\in[0,i]}$ and the fusion center's decision $\{y_j\}_{j\in[0,i]}$. As the LEB attacker has no initial knowledge of the decision model O, the learning step of the LEB attack should be generic and flexible. Our learning idea is inspired by *no free lunch theorems* and *transferability*.

The basic idea of all versions of *no free lunch theorems* presented in [53] is that for both deterministic and stochastic algorithms, "the average performance of any pair of learning algorithms across all possible problems is identical", or more explicitly, it demonstrates that "what an algorithm gains in performance on one class of problems is necessarily offset by its performance on the remaining problems".

No free lunch theorems imply that learning algorithms perform well on one problem does not mean it can always perform well when the problem changes, and that it is not a good idea to use a fixed learning model given different real-world scenarios. Therefore, we adopt a "horses for courses" strategy rather than a "once for all" strategy when choosing the surrogate model. We employ a set of learning algorithms as sub-models to comprehend

varieties at the fusion center. At each timeslot, we choose the learning algorithm that has the best performance as the final surrogate model.

Transferability shows that as long as models are trained to perform the same task, the influence of adversarial samples for one model can often be transferred to other models, even if they have different architectures or are trained on different training parameters or datasets [24]. Transferability property offers the justification of using adversarial samples generated from one sub-model to cheat other sub-models.

As a result, we are motivated to use a set of different machine learning models together to approximate the decision model O. Specifically, the surrogate model S_i consists of L machine learning models, called sub-models. Each sub-model is denoted as

$$S_{l,i} : \mathcal{A} \to \mathcal{Y}, \mathcal{A} \subset \mathbb{R}^{m \times 1}, \mathcal{Y} = \{-1, 1\}, l = 1, 2, \cdots, L, \tag{2.2}$$

where m is the number of malicious nodes controlled by the attacker. The number of sub-models L depends on the attacker's resource and strength. When L is large, the attacker will have more options in choosing the best surrogate model. Each sub-model is a particular representative machine learning model (e.g., Support Vector Machine (SVM), linear regression, multi-layer neural network (MLP), etc.) and is trained on the same set $\{< \mathbf{a}_j, y_j >_{j \in [0,i]}\}$.

2.2.2.2 Evaluation

As shown in Figure. 2.3, even we use all of the malicious nodes together to build a surrogate model, the model is only a partial one because the fusion center will also use sensing data from other innocuous nodes, which is unknown to the attacker. Thus, the goal of the evaluation procedure is to (i) evaluate whether the partial model (sub-models in the surrogate model) is accurate enough and (ii) use model selection to select the best sub-model to launch the attack.

At timeslot i, the attacker uses each of the sub-model $S_{l,i}$ to classify the current data vector $\mathbf{a}_i$ and obtains its local decision $z_i = S_{l,i}(\mathbf{a}_i)$. Then the attacker compares z_i with the fusion center's decision y_i, and maintains a metric called internal accuracy for each sub-model $S_{l,i}$, which is defined as $A_{l,i} = \frac{1}{i}\sum_{j=0}^{i} \mathbf{1}_{\{z_j=y_j\}} \in [0,1]$, where $\mathbf{1}_{\{z_j=y_j\}}$ denotes the indicator function that has value of 1, if $z_j = y_j$, and value of 0 otherwise. The internal accuracy measures how accurate a sub-model can track the fusion center's decision model.

At the evaluation step, if the highest internal accuracy in all sub-models is greater than a given threshold α, the attacker will select the sub-model that has the highest internal accuracy as its final surrogate model for timeslot i and enters the beating procedure for the ith timeslot. Otherwise, the attacker will not attack, but simply send the real sensing data to the fusion center. This evaluation procedure ensures that the attacker will not attack with low confidence, which is also designed to improve the attack success probability under trust-based defenses.

We denote by $S_{l^*,i}$ the best sub-model with the highest internal accuracy selected at timeslot i, where $l^* = \arg\max_{l\in[1,L]} A_{l,i}$. Then we formulate the surrogate model as

$$S_i = S_{l^*,i} = S_{\arg\max_{l\in[1,L]} A_{l,i},i}. \tag{2.3}$$

2.2.2.3 Beating

In the learning and evaluation steps, the attacker passively listens and builds its model, and does not enter the beating step and launch any active attack unless it passes the threshold test in the evaluation step. The goal of the beating step is for the attacker to craft adversarial sensing results based on the selected sub-model to beat the decision model at the fusion center and get a desired output. Attacker's adversarial sensing results are denoted as $\mathbf{a}_i^*$. We write $\mathbf{a}_i^* = \mathbf{a}_i + \boldsymbol{\delta}_i$, where $\mathbf{a}_i$ is the real sensing data and $\boldsymbol{\delta}_i$ is the adversarial perturbation.

The beating step finds the best $\boldsymbol{\delta}_i$ with the minimum data perturbation satisfying

$$\begin{aligned} &\text{Objective:} && \arg\min_{\boldsymbol{\delta}_i \in \mathcal{D}} \|\boldsymbol{\delta}_i\|_2, \\ &\text{Subject to:} && S_{l^*,i}(\mathbf{a}_i^*) = S_{l^*,i}(\mathbf{a}_i + \boldsymbol{\delta}_i) \neq S_{l^*,i}(\mathbf{a}_i), \end{aligned} \tag{2.4}$$

where $\|\cdot\|_2$ is the L2-norm and $\mathcal{D}$ is the feasible solution space of $\boldsymbol{\delta}_i$, which is a constraint put onto $\mathbf{a}^*$ to limit the maximum report $\mathbf{a}_{\max}$ and minimum report $\mathbf{a}_{\min}$, which could alleviate the risk of being detected by defenses like outlier-based malicious detection [12]. As the objective is to minimize the perturbation of given inputs, we found from real-world experiments that it can also alleviate much of other Byzantine detection methods [10, 19].

We note that the above optimization objective does not always guarantee a solution as a result of restriction from feasible solution space $\mathcal{D}$. More information about how to generate $\boldsymbol{\delta}$ under restriction $\mathcal{D}$ is detailed in Section 2.2.3.

After the beating process, all sub-models in the LEB attack framework need to be updated continuously over time and will be retrained by adding new input and output data in each round of spectrum sensing, which is the new learning step of the next episode. The nature of such a retraining process is to incrementally add training data to its already trained model, such that a more accurate model can be found over time. Therefore, the LEB attack is designed to employ online/incremental sub-models to efficiently update the surrogate model, such that it can be updated with new sensing data in a dynamic network environment.

2.2.3 Generic Adversarial Sensing Data Generation

As discussed in the beating step, we must find the best $\boldsymbol{\delta}_i$ with the minimum data perturbation to satisfy Eq. (2.4). When the best sub-model is chosen as the surrogate model $S_i = S_{l^*,i}$, $\boldsymbol{\delta}_i$ can be found via solving a method-specific optimization problem. For example, we can use the fast gradient sign methods (FGSM) [54] or the Jacobian-based

saliency map approach (JBSM) [55] to find the best $\boldsymbol{\delta}_i$ if the selected sub-model is a deep neural network. However, this complicates the solution to Eq. (2.4) in the beating step, makes it dependent on a specific type of sub-model, and makes it less flexible and generic as the surrogate model contains L sub-models, which is extendable by design in the LEB attack framework.

We aim to provide a generic adversarial sensing data generation algorithm to solve Eq. (2.4). Our design intuition has two main components: (i) Unlike complicated data representation (e.g., image and voice data), signal strength data provides straightforward information: a larger value more likely indicates that a channel is occupied and a smaller value indicates otherwise. Therefore, searching is biased towards one direction in Eq. (2.4). (ii) Transferability in machine learning indicates that if we can find adversarial examples of one sub-model, we can transfer it to other sub-models.

However, in the LEB attack framework, different final surrogate models will be selected over time. Inspired from that, we propose a pilot model-based method for adversarial sensing result generation. This method is specifically designed for the cooperative spectrum sensing scenario, which consists of two steps: *Step 1,* estimating the decision hyper-plane of training sensing results $\{< \mathbf{a}_j, y_j >_{j\in[0,i]}\}$ from malicious nodes, which is determined by $\mathbf{w}\cdot\mathbf{a}+b=0$ for a small set of sensing results ($\mathbf{a}$ comes from $\{\mathbf{a}_j\}_{j\in[0,i]}$, also known as support vectors in SVM [56]) from the training data of the surrogate model. $\mathbf{w}$ is the trained weight vector and b is the bias (or intercept) (as shown in Figure. 2.4, in which we consider a two-dimension situation, denoted as x_1 and x_2); *Step 2*, using a binary search structure along the direction defined by $\mathbf{w}$ to find the final $\boldsymbol{\delta}_i$ to form the adversarial report $\mathbf{a}^*$.

The procedure of the proposed method is shown in Algorithm 1. We can use SVM as the pilot model due to its strong transferability performance [24] in practice. However, it is not necessary to train an extra pilot model; instead, we can choose the sub-model that has the best consistency in terms of internal accuracy with the fusion center as the pilot model

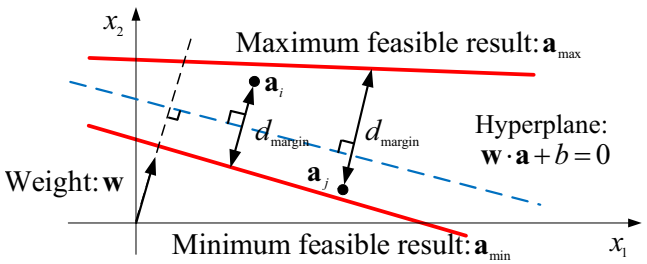


Figure 2.4. Pilot model-based adversarial sensing result generation.

Algorithm 1 Adversarial Sensing Result Generation

Input: Sensing result $\mathbf{a}_i$, feasible solution space $\mathcal{D}$, selected sub-model S_i, algorithm termination threshold ϵ, trained pilot model parameters: $\mathbf{w}, b$;
Output: Adversarial perturbation $\boldsymbol{\delta}_i$;

Set two boundary vectors $\mathbf{a}_{\min}$ and $\mathbf{a}_{\max}$ according to $\mathcal{D}$;
Set $sgn = -sign(\mathbf{w} \cdot \mathbf{a}_i + b)$;
Set initial value: $\boldsymbol{\delta}_i \leftarrow sgn \times d_{\text{margin}}\mathbf{w}$;
if $S_i(\mathbf{a}_i + \boldsymbol{\delta}_i) \neq S_i(\mathbf{a}_i)$, **then**
  $l \leftarrow 0$; $r \leftarrow d_{\text{margin}}$;
  repeat
    if $S_i(\mathbf{a}_i + (sgn(l+r)/2)\mathbf{w}) \neq S_i(\mathbf{a}_i)$, **then**
      $r \leftarrow l + (r-l)/2$;
    else
      $l \leftarrow l + (r-l)/2$;
    end if
  until $\|r - l\| \leq \epsilon$;
  if $S_i(\mathbf{a}_i + sgn \times r\mathbf{w}) \neq S_i(\mathbf{a}_i)$, **then**
    Return the adversarial perturbation:
    $\boldsymbol{\delta}_i \leftarrow sgn \times r\mathbf{w}$.
  else
    terminate without any feasible solution.
  end if
else
  terminate without any feasible solution.
end if

among models that involve learning a hyper-plane (such as Passive Aggressive algorithm [57] and other classifiers). Because of the logarithmic complexity ($O(logN)$ with searching size N) of the binary search structure, the proposed algorithm is expected to yield a fast and efficient solution to Eq. (2.4) in cooperative spectrum sensing scenarios.

2.3 Influence-Limiting Defense

The LEB attack poses a new security threat to cooperative spectrum sensing. In this section, we propose a countermeasure policy named influence-limiting defense to combat LEB attacks. We design the influence-limiting defense as a non-invasive policy such that it can coexist with traditional defenses [10, 11, 12, 14, 19].

2.3.1 Defense Motivation

Many traditional defenses such as statistics-based defenses [12, 17, 19], machine learning-based defenses [14, 21, 10, 22] and trust value-based methods [13, 11, 58] have been designed by making assumptions about attackers. In a scenario where learning-empowered attackers are present, there exist following concerns:

- The identified model mismatch phenomenon that results from spatial and temporal unevenness will lead to dynamic changes of the statistical property at each sensing node.

- When the attacker decides not to launch the attack, it can make controlled nodes behave "normal" by correcting random mistakes and combining all the sensing results of manipulated nodes to make the channel status decision, which improves the statistical consistency of those controlled nodes with the fusion center at timeslots without attacks, such that they can maintain a competitive attack budget at timeslots with attacks.

- When the manipulated nodes decide to launch the attack, they can learn an efficient way through the LEB attack framework, which minimizes the pattern deviations.

Although many traditional methods have been proposed to counter uncertainties of sensing results from each node (e.g., metrics of "weight", "trust value", and "reputation" are

widely used to balance the individual decision of each node and the fusion center decision [59, 9, 10, 11, 12, 13, 14, 15, 16]), most of these metrics are directly or indirectly based on the statistical consistencies of individual decisions with global decisions. Nevertheless, high consistencies do not guarantee high worthiness of trust. As we have detailed in the LEB attack strategy, malicious nodes can be controlled to purposely maintaining a high consistency with the fusion center to avoid being detected.

To our best knowledge, robust and generic strategies to combat these challenges in cooperative spectrum sensing are largely missing. We focus on finding the reason why a limited number of malicious nodes can succeed after learning the fusion center's model. Our key observation is that malicious nodes can take advantage of the model mismatch phenomenon to build up their dominant roles in the decision inference process, i.e., malicious nodes can accumulate their influence on the fusion center, thus the influence of normal nodes are comparatively decreased. From the defense perspective, although we may not know identities of attackers in the network, we can limit the influence of any subset of nodes on the fusion center's global decision.

Our objective is to design a robust and generic countermeasure named influence-limiting defense specifically for the spectrum sensing domain. The method aims to bridge the gap between traditional defense methods and new challenges posed by LEB attacks by introducing the decision-flipping influence.

2.3.2 Defense Framework

We detail the influence-limiting defense framework in the following subsections.

2.3.2.1 Defense Objective

To measure the influence of any given subset of nodes $\mathcal{X}_{\text{sub}}$, we propose a new metric called *decision-flipping influence*, denoted as $I(\mathcal{X}_{\text{sub}})$,

$$\begin{aligned} I(\mathcal{X}_{\text{sub}}) &= \frac{\#\text{ of } \mathbf{a}^*}{\#\text{ of } \mathbf{a}}, \\ \text{s.t. } & O(\mathbf{x}^*) \neq O(\mathbf{x}), \end{aligned} \tag{2.5}$$

in which $\mathbf{a}^*$ is from $\mathcal{X}_{\text{sub}}$, and is part of $\mathbf{x}^*$. Intuitively, (2.5) indicates the probability of finding a malicious input $\mathbf{x}^*$ that flips the decision output given $\mathbf{x}$ by changing $\mathbf{a}$. Apparently, the influence of all nodes is 1, i.e., $I(\mathcal{X}) = 1$. Unlike "weight", "trust value" or "reputation" based measures and the influence-based measure proposed in [60, 61], the decision-flipping influence $I(\mathcal{X}_{\text{sub}})$ is a direct probabilistic measure of the role $\mathcal{X}_{\text{sub}}$ played in the final decision process.

Starting from the decision-flipping influence $I(\mathcal{X}_{\text{sub}})$, we can mitigate the severe impact of potential LEB attacks by enforcing an influence-limiting policy, in which the decision-flipping influence of any subset $\mathcal{X}_{\text{sub}} \subset \mathcal{X}$ should satisfy $I(\mathcal{X}_{\text{sub}}) \leq \delta(|\mathcal{X}_{\text{sub}}|)$, where $|\mathcal{X}_{\text{sub}}|$ denotes the number of nodes in $\mathcal{X}_{\text{sub}}$, and $\delta(|\mathcal{X}_{\text{sub}}|)$ is the threshold function of $|\mathcal{X}_{\text{sub}}|$, i.e., the influence-limiting policy is triggered only when $I(\mathcal{X}_{\text{sub}})$ goes beyond $\delta(|\mathcal{X}_{\text{sub}}|)$. For example, $\delta(1)$ denotes the threshold to limit the influence of each individual node.

From the LEB attack framework, we know that as malicious nodes keep succeeding to flip the decision of the fusion center, their corresponding decision-flipping influence $I(\mathcal{X}_{\text{sub}})$ will also increase, thus influence-limiting policy will be triggered. We can write the influence-limiting policy as follows:

$$\begin{aligned} \text{Objective:} \quad & \text{minimize } (y - \hat{y})^2, \\ \text{Subject to:} \quad & I(\mathcal{X}_{\text{sub}}) \leq \delta(|\mathcal{X}_{\text{sub}}|), \ \ \forall \mathcal{X}_{\text{sub}} \subset \mathcal{X}, \end{aligned} \tag{2.6}$$

in which y is the true channel status and $\hat{y}$ denotes the predicted decision output. This is a non-invasive method that can coexist with traditional/existing defenses as mentioned in Section 2.2.

2.3.2.2 Finding Threshold Function $\delta(|\mathcal{X}_{sub}|)$

The threshold function of $\delta(|\mathcal{X}_{\text{sub}}|)$ with regard to $|\mathcal{X}_{\text{sub}}|$ is the core component of the influence-limiting defense. To choose the threshold function, we first discuss simple cases, then extend them to generic cases:

(i) In a well-balanced cooperative spectrum sensing scenario without malicious nodes, the value of $\delta(|\mathcal{X}_{\text{sub}}|)$ in terms of $|\mathcal{X}_{\text{sub}}|$ intuitively needs to satisfy the following three basic requirements: (a) $\delta(|\mathcal{X}_{\text{sub}}|) \to 0$ when $|\mathcal{X}_{\text{sub}}| \to 0$; (b) $\delta(|\mathcal{X}_{\text{sub}}|)$ is monotonically increasing with regard to $|\mathcal{X}_{\text{sub}}|$; (c) $I(\mathcal{X}_{\text{sub}}) \to \frac{1}{2}$ when $|\mathcal{X}_{\text{sub}}| \to \frac{n}{2}$, which is to ensure that the influence over the fusion center is dominated by the majority rather than a small group of sensing nodes.

Based on the requirements mentioned above, a sigmoid style function comes out to be a feasible choice to interpolate the function of $\delta(|\mathcal{X}_{\text{sub}}|)$ with regard to $|\mathcal{X}_{\text{sub}}|$,

$$\delta(|\mathcal{X}_{\text{sub}}|) = \frac{1}{1 + e^{-c_1(|\mathcal{X}_{\text{sub}}| - \frac{n}{2})}}, 0 \leq |\mathcal{X}_{\text{sub}}| \leq \frac{n}{2}, \tag{2.7}$$

where c_1 is the control parameter used to adjust the function to better interpolate various practical scenarios. We assume that the number of malicious nodes is less than the number of normal nodes, which is the typical attack model [9, 10, 12, 15, 16].

(ii) In a generic scenario where malicious nodes may present, the function $\delta(|\mathcal{X}_{\text{sub}}|)$ defined in (i) has no mechanism to counter malicious nodes. It is intuitive that when malicious nodes are present in $\mathcal{X}_{\text{sub}}$, $\delta(|\mathcal{X}_{\text{sub}}|)$ should be limited to a smaller threshold value such that

the decision-flipping influence can be restrained. The next question is how one can know which node is malicious.

Note that due to the temporal or spatial unevenness, it is difficult to accurately identify which node is indeed malicious by checking the statistical property of the signal strengths. Therefore, instead of offering a hard decision rule to clearly classify a node into either innocuous or malicious, we design a soft rule to discriminate certain nodes in the final decision by the fusion center.

In particular, we still leverage a node's signal strength, but only checking changes of its statistical property. Suppose when a node's signal strengths exhibit different properties during the training and testing (or decision) phases, there exist following indications:

- The node may be malicious and its signal strengths are manipulated for an effective attack. If this is the case, the node should be at least less weighted (if not eliminated) in the fusion center's decision.

- The node may be legitimate but its signal property changes due to the temporal or spatial unevenness, which further means that the original training data for this node does not reflect its current signal property and thus becomes less useful for the current decision.

In both cases, we should at least weight those nodes less in the final decision. Therefore, we adopt the Kolmogorov-Smirnov (K-S) statistic to quantify such a change for a node. For node j, the K-S statistic d_{ks}^j is

$$d_{\text{ks}}^j = \sup_x |F_T^j(x) - F_C^j(x)|, \tag{2.8}$$

where $F_T^j(x)$ is the training data distribution for node j, $F_C^j(x)$ is the empirical distribution of node j's signal strength in the test period, representing its current signal property, and $\sup(\cdot)$ denotes the maximum value or upper bound.

$F_T^j(x)$ and $F_C^j(x)$ can be estimated from the sensed data. We can employ kernel density estimation method to obtain the distribution due to its flexibility in choosing kernel functions. Based on Eq. (2.8), we present our generalized influence-limiting policy as

$$\begin{aligned} \delta(|\mathcal{X}_{\text{sub}}|) &= \frac{1}{1+e^{-c_1(|\mathcal{X}_{\text{sub}}|-\frac{n}{2})}} - c_2 \sum_{j\in\mathcal{X}_{\text{sub}}} d_{\text{ks}}^j, \\ 0 \le |\mathcal{X}_{\text{sub}}| &\le \frac{n}{2}, 0 < \delta(|\mathcal{X}_{\text{sub}}|) < 1, \end{aligned} \tag{2.9}$$

where c_1 is a cost control parameter and c_2 is an influence control parameter.

The policy ensures that if there are more abnormal nodes in $\mathcal{X}_{\text{sub}}$, then $\sum_{j\in\mathcal{X}_{\text{sub}}} d_{\text{ks}}^j$ will be larger and thus the threshold $\delta(|\mathcal{X}_{\text{sub}}|)$ will be smaller. When all the nodes in $\mathcal{X}_{\text{sub}}$ are normal nodes, $\sum_{j\in\mathcal{X}_{\text{sub}}} d_{\text{ks}}^j$ is expected to be small.

2.3.2.3 Enforcing Influence-limiting Policy

At the fusion center, the objective of the defense is to mitigate the impact posed by potentially malicious nodes through enforcing the influence-limiting policy, i.e., limiting the decision-flipping influence of any given $\mathcal{X}_{\text{sub}}$ within the range of $[0, \delta(|\mathcal{X}_{\text{sub}}|)]$.

There are multiple ways to enforce the policy. One way is to re-weight the sensing results of $\mathcal{X}_{\text{sub}}$. Specifically, when the measured decision-flipping influence $I(\mathcal{X}_{\text{sub}})$ goes beyond $\delta(|\mathcal{X}_{\text{sub}}|)$ in a traditional defense method, the influence-limiting policy will limit the weight of $\mathcal{X}_{\text{sub}}$ to the threshold. In other words, the weight $w(\mathcal{X}_{\text{sub}})$ can be written as

$$w(\mathcal{X}_{\text{sub}}) = \begin{cases} \delta(|\mathcal{X}_{\text{sub}}|), & I(\mathcal{X}_{\text{sub}}) \ge \delta(|\mathcal{X}_{\text{sub}}|), \\ \text{Original weight}, & \text{otherwise.} \end{cases} \tag{2.10}$$

Algorithm 2 Influence-limiting defense

Input: Historical sensing results $\mathcal{X}$ and the corresponding decision outputs in $\mathcal{Y}$; parameters c_1, c_2, η.

for each $\mathcal{X}_{\text{sub}}$ according to the value of η:
Calculate the decision-flipping influence $I(\mathcal{X}_{\text{sub}})$;
Compute the threshold $\delta(|\mathcal{X}_{\text{sub}}|)$;
Enforce influence through limiting the weight $w(\mathcal{X}_{\text{sub}})$;
end for
Feedback $w(\mathcal{X}_{\text{sub}})$ to traditional/existing defenses.

Another way to enforce this policy is to work with an existing defense such as the incentive method proposed in [62] to re-weight the corresponding nodes, thus the suspicious nodes can be discouraged and suppressed.

2.3.2.4 Balancing Effectiveness and Complexity

It is worth noting that in order to achieve the objective defined in Eq. (2.6), we have to consider all possible subsets in $\mathcal{X}$. However, it will be computationally cumbersome to enforce the full influence-limiting policy on a cooperative spectrum sensing network with a large number of nodes, as the total number of subsets in $\mathcal{X}$ with n nodes is $\sum_{k=1}^{n} \binom{n}{k}$.

We provide a practical strategy to reduce the complexity of the influence-limiting policy. We introduce a parameter $\eta, 0 < \eta \leq n$, in the influence-limiting policy and consider limiting the influence of any subset which has no more than η nodes. For example, $\eta = 1$ means that we perform the influence-limiting policy only on each individual node inside the network and $\eta = n$ means a full scale influence-limiting policy evaluated on all possible combinations of $\mathcal{X}_{\text{sub}}$. By adjusting the value of η, we can balance the complexity of the influence-limiting policy and the effectiveness against potential attacks.

The step-by-step process of the influence-limiting defense can be seen in Algorithm 3. The advantage of using influence-limiting policy is that we can add this enforcement as a parallel, non-invasive constraint to the primary/traditional decision method at the fusion

center. The influence-limiting policy is applied to bridge the gap between traditional defenses and the new adversarial machine learning-based attacks, such as LEB attacks.

2.4 Low-cost Influence-Limiting Defense

In this section, given the high computation cost of the original influence-limiting defense, we provide a low-cost version of the influence-limiting defense method.

In the influence-limiting defense, the core idea is to contain or limit the decision-flipping influence of $\mathcal{X}_{\text{sub}}$ towards the decision process in the fusion center. However, in practice, to fully limit the attack power of all malicious nodes, it is necessary to enforce the influence-limiting defense at different levels in terms of the size of $\mathcal{X}_{\text{sub}}$. The defender does not usually know how many nodes in its cooperative spectrum sensing network are controlled by the attacker. Note that the total number of subsets in $\mathcal{X}$ with n nodes is $\sum_{k=1}^{n} \binom{n}{k}$, which means the computational complexity in terms of subset will be at the level of $O(2^n)$. This might make it difficult for the fusion center to enforce a full version of influence-limiting defense. Therefore, another parameter η was introduced in [63] to control the complexity.

From a practical point of view, we do not need to evaluate all the combinations of $\mathcal{X}_{\text{sub}}$ from $\mathcal{X}$. For example, if we find that the decision-flipping influence for a specified subset $\mathcal{X}_{\text{sub}}$ is below the value defined by the cap function, then we actually do not need to further examine all the subsets within $\mathcal{X}_{\text{sub}}$, which will reduce the computational cost of enforcing influence-limiting defense significantly. Therefore, the problem turns out to be a challenge to find the most suspicious subset of $\mathcal{X}_{\text{sub}}$ to enforce the full version of influence-limiting policy within that subset. Inspired from the "divide and conquer" scheme in traditional algorithm design, we propose a low-cost version of influence-limiting defense.

We employ the strategy of a top-down style, in which we first divide all the sensing nodes into two subsets randomly and then evaluate the decision-flipping influence $I(\mathcal{X}_{\text{sub}})$ of each subset. In a normal scenario where no malicious nodes are present and each innocuous node

is also well-behaving, the decision-flipping influence for each subset should be very similar with each other and well-balanced. However, if there exist malicious nodes in either subset or in both subsets, the decision-flipping influence for the subset that has the larger number of malicious nodes will have a higher decision-flipping influence if the attacker is successful. If our cost control parameter c_1 and influence control parameter c_2 are chosen appropriately, the proposed influence-limiting defense should be triggered on that subset. Then, in the next round of search, we focus on the subset that triggers the influence-limiting defense. We perform the second round of "divide" operation on the subset, divide it into two smaller subsets and evaluate the decision-flipping influence of each subset again. We will continue the search iteratively by dividing the chosen subset.

At the end of "divide and conquer" search process, we will reach a point where both the two subsets trigger the influence-limiting defense, or at least one subset is non-divisible if there exist malicious nodes like LEB attackers. There is a possibility that both of the subsets include enough malicious nodes to make their decision-flipping influence larger than the value defined by the cap function $\delta(|\mathcal{X}_{\text{sub}}|)$. If there is no malicious node, it will be very difficult to trigger the defense based on the definition of $\delta(|\mathcal{X}_{\text{sub}}|)$.

The key point of this "divide and conquer" strategy is that we do not need to enforce the influence-limiting defense until we reach the point where both the subsets trigger the defense. Before that point, all we need to do is to evaluate the decision-flipping influence $\mathcal{I}(\mathcal{X}_{\text{sub}})$, which will decrease the computational cost of enforcing influence-limiting defense significantly. The detailed process of our low-cost version of influence-limiting defense is shown in Algorithm 3. The advantage of the proposed low-cost version of influence-limiting defense is that it divides the set of the sensing nodes evenly (or roughly evenly if the number of nodes is an odd number) into two subsets. Thus, it will be very efficient to locate the subset that reaches the cap function if there are malicious nodes.

2.5 Experimental Results and Analysis

In this section, we present our real-world dataset collected from practical TV white space signal strengths. Based on this dataset, we conducted experiments to measure the impact of LEB attacks under various conditions.

2.5.1 LEB Attack Experimental Setup

We present measurement details, configurations of the fusion center, the LEB attack framework, and performance metrics in this subsection.

2.5.1.1 Measurement Configurations

We collected realistic TV white space signal strengths using RTL-SDR TV dongles, which have been validated to have adequate signal detection capabilities [48]. We deployed 20 RTL-SDR TV dongles as sensing nodes on a campus area to collect signal strengths simultaneously on TV channels based on configurations used in [48]. We used GNURadio v3.7.9 to implement the sensing process for each dongle that uses the averaged signal power over a time period of 30 seconds (required by Federal Communications Commission [64]) as the sensing result for one timeslot.

To make the collected signal data comprehend the realistic spectrum sensing scenarios as much as possible, we deployed sensing devices in various surrounding environments. Figure. 2.5 shows how 20 RTL-SDR TV dongles were deployed throughout a 379×232 ft^2 building: 8 TV dongles were placed outside of the building and 12 were distributed within the building on different floors and indoor environments. We distributed TV dongles at these various places and environments to represent different spectrum sensing scenarios in practice, which is similar with the deployment style of CORNET testbed [65]. We collected

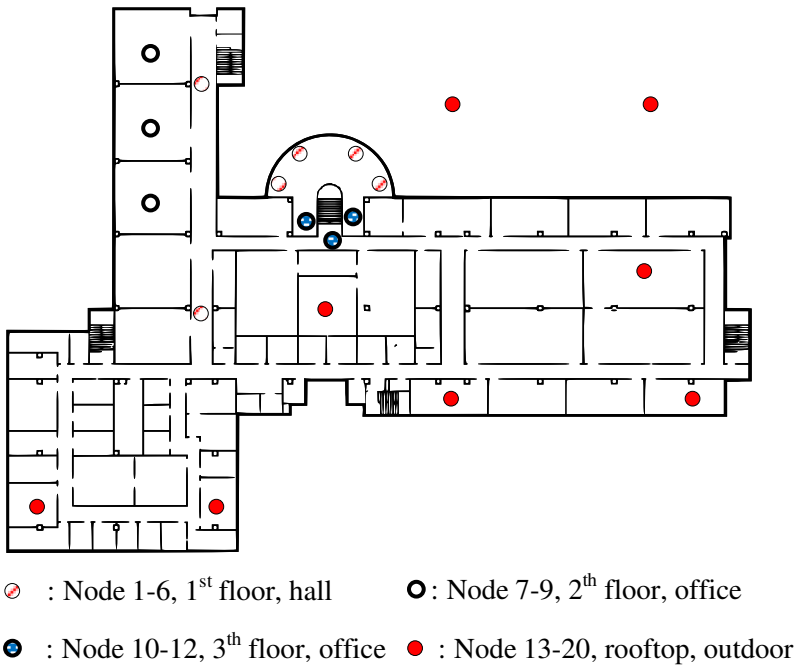


Figure 2.5. The deployment environment of 20 RTL-SDRs.

signal strengths on 22 different channels for 100 continuous hours on each sensing node, such that the dynamic model mismatch phenomenon can be recorded in the dataset.

2.5.1.2 Fusion Center Configurations

To show the attack performance of the LEB attack framework, we implemented eight existing representative intrusion detection defenses at the fusion center. Specifically, four statistics-based defenses: Outlier factors based defense (Outlier) [12], Local Outlier Factor based defense (LOF) [66], Empirical Covariance based (EmpCov) and Robust Covariance based (RobCov)) detections [67]; three machine learning based defenses: fuzzy kNN based defense (fzKNN) [21], Double-Sided Neighbor Distance based defense (DSND) [10] and One-class SVM based detection (OCSVM) [22]; and one trust-based detection method (Trust) [11].

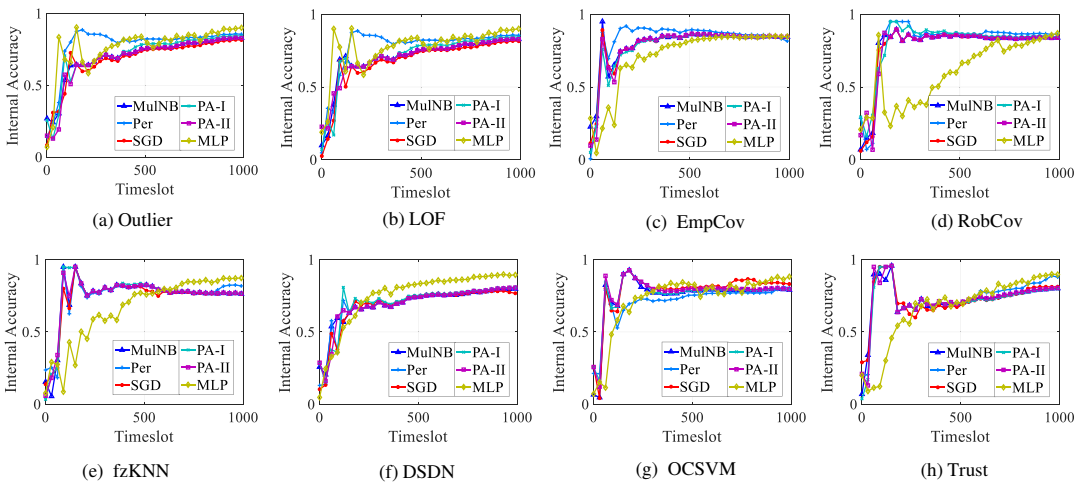


Figure 2.6. Internal accuracy transitions of the LEB attacker under different defenses.

There are various data fusion rules such as SVM, Logistic Regression (LR), AND, OR and Majority rule[61, 66, 14, 68]. Due to the LEB attack design and the property of transferability in machine learning, the fusion rule does not generally affect the design of the LEB attack architecture, but may affect its attack result. For example, the majority rule may perform better than SVM against the LEB attack (as the attack is likely to fail when malicious nodes are not the majority) but it does not generate good performance as pointed out in the literature [14, 69]. We choose the SVM rule as the representative rule in our experimental evaluation because (i) it is one of the most widely used fusion rules for efficiency and performance [14, 68], and (ii) our main goal is to show that how a machine learning powered attacker can take advantage of the open nature of cooperative dynamic spectrum sensing to launch a new type of attack.

We used the sensed data of the first five hours collected from 22 nodes and the corresponding ground truths of the TV channel status in the local area to build statistical models and as training data for the defense methods implemented at the fusion center. The remaining data was used as test data.

2.5.1.3 LEB Attack Framework Configurations

We implemented the surrogate model based on manipulated nodes, which consists of six incremental learning based sub-models: Naive Bayes classifier for multinomial models (MulNB), Perceptron classifier (Per), Linear SVM classifier with stochastic gradient descent training (SGD), Passive Aggressive-I classifier (PA-I), Passive Aggressive-II classifier (AP-II) and Multi-layer Perceptron classifier (MLP). These six models are well-known and representative techniques in incremental learning. In practice, we can add other incremental learning-based algorithms or exclude models based on security requirements of specific scenarios.

Without loss of generality, we let the interaction of cooperative spectrum sensing and the LEB attacker that controls the malicious nodes start at timeslot 0 corresponding to the 6th hour of the collected data. We assume that the defense mechanism has already been trained using the first five hours of the collected data. At each timeslot that takes 30 seconds, the LEB attacker learns and evaluates the surrogate models to launch a potential attack.

In the evaluation step, the internal accuracy threshold α is 0.85 unless otherwise specified. In the beating step, the LEB attacker generates adversarial sensing data based on the generic generation algorithm proposed in Section 2.2.3.

2.5.1.4 Performance Metrics

We evaluated the performance of the LEB attacker through two metrics: *attack success ratio* and *overall disruption ratio*. We define the attack success ratio as the ratio of the number of attacks that successfully mislead the fusion center to make wrong decisions to the number of attack attempts. We define the overall disruption ratio as the ratio of the number

of successful attacks to the number of elapsed timeslots, i.e.

$$\begin{aligned} \text{attack success ratio} &= \frac{\#\text{of successful attacks}}{\#\text{of attack attempts}}, \\ \text{overall disruption ratio} &= \frac{\#\text{of successful attacks}}{\#\text{of elapsed timeslots}}. \end{aligned} \tag{2.11}$$

Note that a higher attack success ratio does not necessarily mean a higher overall disruption ratio. The reason is that when the LEB attacker does not pass the evaluation step (e.g., it may have a large threshold α to make an attack attempt succeed with high probability), it will not launch the attack and thus will not cause a disruption to the network. In comparison, the attack success ratio measures the learning and evaluation quality of the LEB attack framework, and the overall disruption ratio quantifies the performance impact that the LEB attacker brings to the entire cooperative sensing network.

2.5.2 LEB Attack Results and Analysis

We show experimental results of the LEB attack in terms of attack success ratio and overall disruption ratio measurements in this subsection.

2.5.2.1 Attack Impacts on Defense Strategies

We first evaluated the impact of the LEB attacker on each of the 8 defense strategies used by the fusion center. We randomly selected 8 nodes as malicious nodes to be controlled by the LEB attacker. The LEB attacker updates its internal accuracies for each sub-model to evaluate and beat the fusion center. As we do not assume any specific statistical/stationary model for the behavior of the attacker and nodes, sensing reports, or wireless spectrum properties, it can be mathematically infeasible to accurately formulate the interactions between the attacker and the fusion center or perform analytical convergence analysis. Our experimental evaluation results in Figure. 2.6 shows the attacker's internal accuracies and

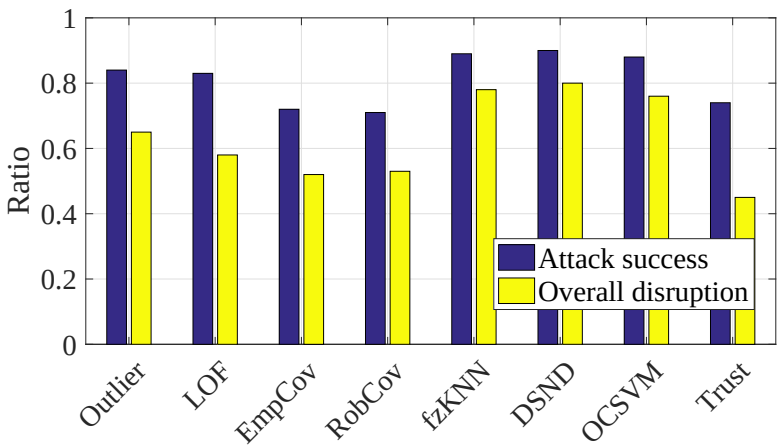


Figure 2.7. The performance of the LEB attacker with different defense strategies.

the convergence trends of sub-models with regard to the timeslot against defense strategies (a) Outlier, (b) LOF, (c) EmpCov, (d) RobCov, (e) fzKNN, (f) DSDN, (g) OCSVM and (h) Trust.

We observe from Figure. 2.6 that when the attacker starts to learn, internal accuracies of each sub-model change drastically; but with more timeslots elapsed, they gradually become stable. For example, in Figure. 2.6 (c), accuracies start to stabilize at around 0.85 after 500 timeslots, which is the approximated convergence time in the training process of the surrogate model.

Figure. 2.7 illustrates the attack success ratio and overall disruption ratio of the LEB attacker against each defense method individually. From Figure. 2.7, we note that the LEB attacker achieves 71%–90% attack success ratios against different methods, which means that the learning and evaluation design in the LEB attack framework is effective for the attacker to assess its potential capability to launch successful attacks. We also observe from Figure. 2.7 that the overall disruption ratio is 45%–80% due to the attack, which indicates that the LEB attacker is able to successfully fool the fusion center into making wrong decisions for 45%–80% of the time, thereby resulting in severe performance disruption.

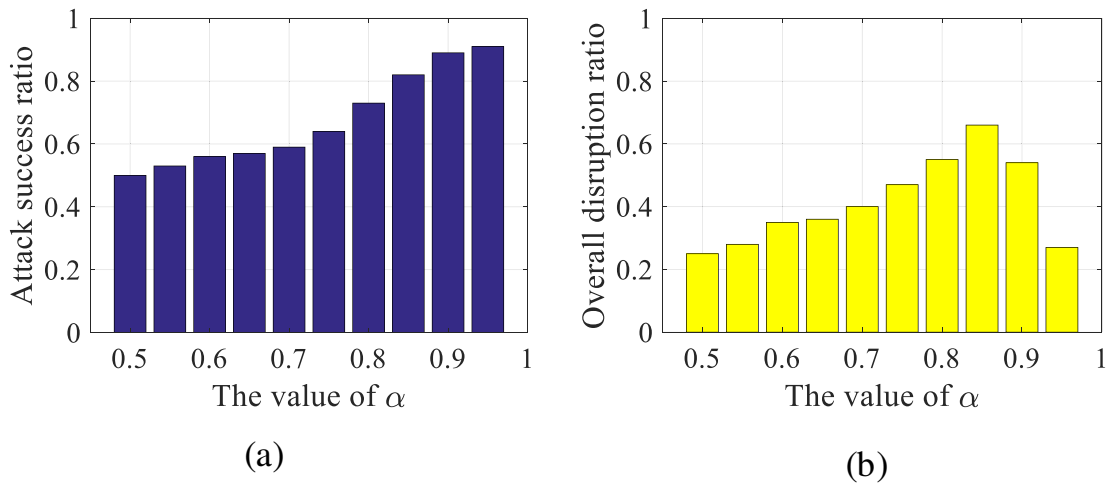


Figure 2.8. The relationship between α and the attack success ratio of the LEB attacker.

2.5.2.2 Impacts of Threshold α

A key factor in the LEB attack framework is the internal accuracy threshold α. The attacker can only launch attacks if the internal accuracy of a sub-model exceeds α. A larger α should lead to a higher attack success ratio but should also decrease the overall disruption ratio, which is because a larger α will decrease attack attempts but result in a better estimation about the decision rule.

Figure. 2.8 depicts (a) the attack success ratio and (b) the overall disruption ratio for different values of α (the rest of the setting is the same as that in Figure. 2.7 and results are averaged over eight defense strategies). We observe from Figure. 2.8(a) that when α is 0.7 or less, the attack success ratio is around 0.5. As α increases, the attack success ratio also increases. The reason is that an increased α requirement indicates a higher consistency or better estimation of the fusion center for the selected sub-model.

However, in terms of the overall disruption ratio, although a smaller α will lead to more attack attempts, it will also increase the risk for the attacker to activate defense mechanisms at the fusion center, further decreasing the overall disruption ratio. The overall disruption ratio increases when α is small and then decreases as the threshold α approaches to 1, as

Table 2.1. Overall disruption ratios (%) under different scenarios.

Defenses	**The number of malicious nodes m**								
	2	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Outlier	8	20	23	24	34	38	65	68	72
LOF	6	15	23	25	35	36	58	60	65
EmpCov	9	22	27	35	37	39	52	59	66
RobCov	7	15	22	30	39	41	53	56	64
fzKNN	10	21	27	45	45	49	78	80	82
DSND	11	23	31	44	46	47	80	85	86
OCSVM	9	21	31	39	48	51	76	79	80
Trust	6	16	21	29	34	38	45	56	60
Average	8	19	28	34	40	42	63	69	72

shown in Figure. 2.8(b). The reason is that as the threshold α approaches to 1, the number of attack attempts decreases significantly. Thus, although the attack success ratio increases, the overall number of attacks decreases. We observe from our experiments that the optimal threshold can be achieved at around $\alpha = 0.85$ to maximize the overall disruption ratio.

2.5.2.3 Impacts of the Number of Malicious Nodes

The number of malicious nodes controlled by the LEB attacker is a fundamental factor that affects the attack impact, which answers the question of how many nodes an attacker needs to achieve a promising attack performance. Table 2.1 shows the overall disruption ratio against different defenses when the number of randomly selected malicious nodes increases from 2 to 10 (i.e., 10% to 50% of all nodes). Table 2.1 shows that as the number of malicious nodes approaches 10, which is half of the total sensing nodes, the overall disruption ratio (averaged over all defenses) reaches 72%, which means that the spectrum sensing is disrupted by the attacker at 72% of the elapsed timeslots. The attack impact can still be observed even when the number of malicious nodes is small. For example, three malicious nodes (15% of all nodes) can lead to a nearly 20% overall disruption ratio.

It can be concluded from Table 2.1 that the LEB attack framework provides an effective attack strategy even when the number of malicious nodes is small. The attack performance also improves as the number of malicious nodes increases.

2.5.2.4 Impacts of Locations of Malicious Nodes

In previous experiments, we always randomly selected nodes as malicious ones. We are also interested in whether malicious nodes can bring more impact to the network if they choose to be at "better" locations. We divide the dongles into four groups with five in each group. The experiments are conducted by using one group as malicious nodes while the rest being innocuous.

The results are depicted in Figure. 2.9, from which we observe that when controlling nodes 16–20 (distributed in outside environment) as malicious ones, the overall disruption ratio is 43% averaged over all eight defenses. However, if nodes 6–10 (distributed in indoor environment) are controlled as malicious ones, the averaged overall disruption ratio is only 17%. Hence, we can conclude that it is critical for malicious nodes controlled by the LEB attacker to be at "right" locations that have a higher influence on the decision process in order to launch more effective attacks.

2.5.2.5 Efficiency of Adversarial Sensing Data Generation

We also compare the performance of the proposed method for adversarial sensing result generation with other adversarial sample generation methods: Fast gradient sign method (FGSM) [54], Jacobian-based saliency map approach (JBSM) [55], DeepFool method (DF) [70], Basic iterative method (BaIter) [71], SPSA attack method (SPSA) [72] and Elastic-Net method (EN) [73] in Deep Neural Network, which are implemented based on CleverHans V2.1.0 [74].

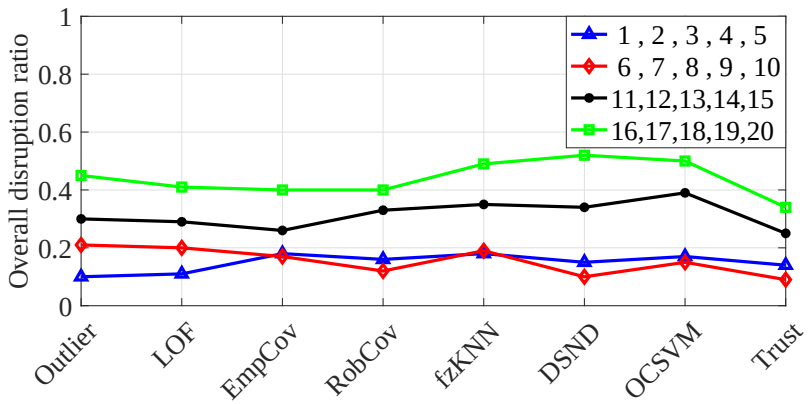


Figure 2.9. Overall disruption ratios when manipulating different sets of nodes.

In these experiments, 8 nodes are randomly selected as malicious nodes and they use different adversarial data generation methods in the LEB attack framework to generate malicious data inputs to the fusion center. Figure. 2.10 shows (a) the attack success ratio and overall disruption ratio under different generation methods and (b) the normalized costs of the generation methods (the normalized cost of a generation method is defined as its computational time to generate the adversarial data vector divided by the computational time of our proposed method to generate the adversarial data; thus our proposed method has a normalized cost of 1).

Figure. 2.10(a) shows that our proposed method has the similar attack success ratio and overall disruption ratio as other methods. Figure. 2.10(b) further shows that our method is much more computationally efficient than the other methods. For example, SPSA has a normalized cost of around 4.5, and the overall average computation cost reduction compared to other methods is around 65%, which is mainly contributed by the simplicity and logarithmic complexity of binary search method in our proposed method.

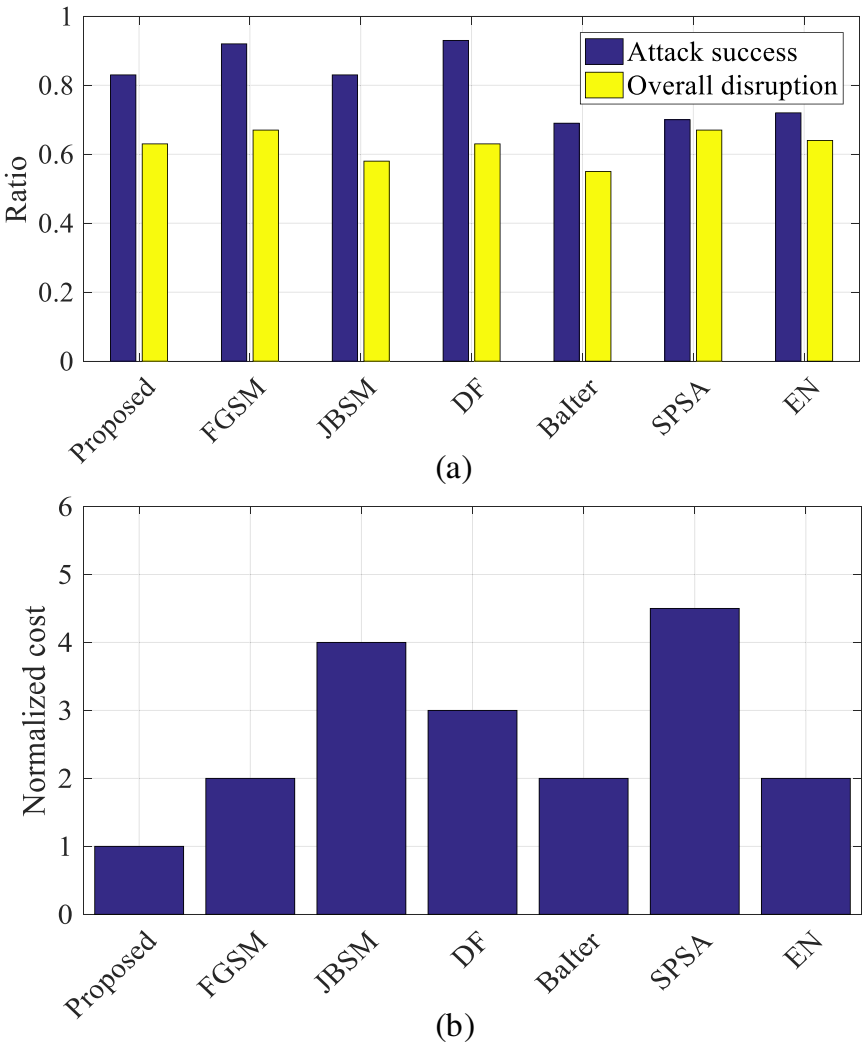


Figure 2.10. Comparison of the adversarial sensing result generation methods.

2.5.2.6 Performance Comparison with Existing Attacks

The LEB attack is a new type of attack against spectrum sensing. There are many other proposed Byzantine attacks against the fusion center. They can be roughly classified into two categories: Independent Malicious Byzantine Attacks (IMBA) and Cooperative Malicious Byzantine Attacks (CMBA) [75, 10, 76]. In IMBA, each Byzantine node makes the decision to attack the fusion center independently, while Byzantine nodes collaborate with each other in CMBA.

IMBA is a comparatively simple attack strategy. It requires no cooperation between the nodes and each individual node makes the decision by itself. It is proved in [75] that unless the number of Byzantines is larger than or equal to 50% of the total nodes, the fusion center cannot be made blind by attackers. More information of the IMBA strategy can be found in [10, 76].

In the CMBA scenario, Byzantine nodes cooperate with each other and make the decision collaboratively to attack the fusion center, which is a much stronger attack model than IMBA. Therefore, the number of malicious nodes required to make the fusion center blind is also less than 50% [75]. LEB attacks belong to the CMBA category, but the attack model is much weaker than the CMBA model in [10]. The LEB attack requires no information of other normal nodes, including the total number of sensing nodes and the sensing results.

In this group of experiments, we compare attack performance of LEB attacks with both IMBA and CMBA methods. Rather than using simple data fusion methods like "AND" [77], "OR" [78], or "Weighted" [79], we employ a more comprehensive machine learning based method, namely SVM as the decision rule [48].

Existing attack methods that we choose for comparison include an independent malicious attack model identified in [10], in which each malicious node launches the attack according to a preset probability; an adaptive probabilistic independent attack model that considers attack costs proposed in [80]; and a cooperative dependent malicious attack model proposed in [81], in which an attacker injects malicious sensing data simultaneously and the falsified data is self-consistent.

Attack success ratios and overall disruption ratios are used to quantify the attack performance. The detailed experimental results are shown in Table 2.2, in which 8 out of 20 nodes are malicious. We can observe that cooperative attacks ([81] and LEB attacks) generally have better performance than independent attacks ([10, 80]), on average more than 10% of improved performance in terms of both attack success ratio and overall disruption ratio.

Table 2.2. Performance comparison under different attack methods.

Attack methods	**Attack success**	**Overall disruption**
Li et al.[10]	0.76	0.76
Ahmadfard et al. [80]	0.73	0.73
Qin et al. [81]	0.84	0.84
LEB attacks	0.95	0.85

Note that unlike the trust-based fusion, the fusion center is trained using SVM. LEB attacks cannot change the values of parameters in the decision model. LEB attacks still have slightly better overall disruption ratio than [81] but they have better attack success ratio (around 10% of the advantage) due to the learning characteristics of LEB attacks.

We conducted comprehensive experiments based on real-world collected spectrum data to validate the LEB attack framework and compared the attack performance with existing defenses and attacks. The principal benefit of the LEB attack framework is due to learning, which supports the attacker to adapt and take advantage of the model mismatch phenomenon in practice.

2.5.3 Influence-limiting Defense Experimental Validation

We evaluated the performance of the proposed influence-limiting defense using the collected dataset. Experimental configuration is remained the same. In this part of experiments, we adopted the trust value-based method as the existing fusion center defense mechanism due to its popularity in cooperative spectrum sensing [13].

2.5.3.1 Impact of c_1 and c_2 for $\delta(|\mathcal{X}_{sub}|)$

We first measured the impact of parameters c_1 and c_2 with regard to the threshold $\delta(|\mathcal{X}_{\text{sub}}|)$ in Eq. (2.9) on the defense performance of the influence-limiting defense.

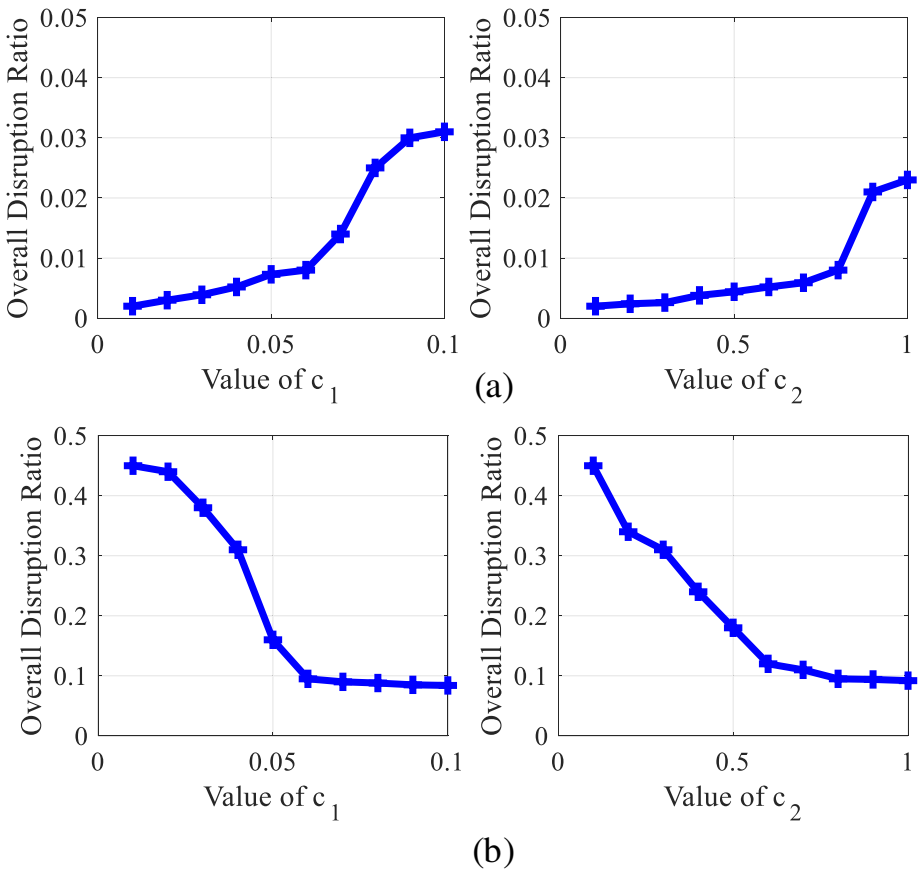


Figure 2.11. The performance when (a) no malicious node exists, (b) LEB attackers exists.

It is obvious that the limitation caused by $\delta(|\mathcal{X}_{\text{sub}}|)$ will lead to a performance cost for the fusion center, especially when there is no malicious node. The cost essentially depends on c_1 in Eq. (2.9). According to the mathematical property of Sigmoid function, less limitation will be enforced when c_1 is smaller; i.e., the cost when no malicious nodes are present will be smaller.

On the other hand, a smaller c_1 hurts the system more when malicious nodes are indeed present. Similarly, a larger value of c_2 makes sure that a potentially malicious node will be penalized more but may also penalize legitimate nodes more when there is no attack. Therefore, we evaluated a wide range of choices of c_1 and c_2 to observe the balance between the performance without the attack and the defense effectiveness in the presence of attacks.

- *Case 1:* When no malicious node exists, all the 20 sensing nodes report the true sensed values to the fusion center. We illustrate the overall disruption ratio in 1000 timeslots with regard to influence-limiting defense of different values for c_1 and c_2 in Figure. 2.11(a). When evaluating one parameter, we fixed the other parameter as the two parameters are independent from each other. We observe from Figure. 2.11(a) that when no malicious node is present, the overall disruption ratio increases as both c_1 and c_2 increase, which corresponds to the slight cost of the influence-limiting defense.

- *Case 2:* When LEB attacks are present, we evaluated the scenario where 8 out of 20 nodes are malicious (i.e., $m = 8$ and $n = 20$). The relationship between different values of c_1 and c_2 and the overall disruption ratio is shown in Figure. 2.11(b). When c_1 or c_2 is very small, the influence limitation is negligible. The defense performance is demonstrated when c_1 and c_2 increase beyond 0.05 and 0.5, respectively.

To make the influence-limiting defense achieve the full potential of defense capability, we choose c_1 and c_2 to decrease the overall disruption ratio when malicious nodes exist, while still maintaining a slight cost when no malicious node is present. Combining the experimental results of the above two cases, we can observe that c_1 and c_2 can be set as 0.05 and 0.5, respectively, in our example.

2.5.3.2 Impact of η on Defense

We also evaluated the defense performance under different values of η that balances the complexity and the defense performance of the influence-limiting policy. We assume $c_1 = 0.5, c_2 = 0.05$ for these experiments. The overall disruption ratio with LEB attacks is shown in Figure. 2.12 when we vary the number of malicious nodes. The results demonstrate that when η is equal or larger than 4, the performance improvement is almost negligible for different cases of $m = 2, 4, 6$ and 8.

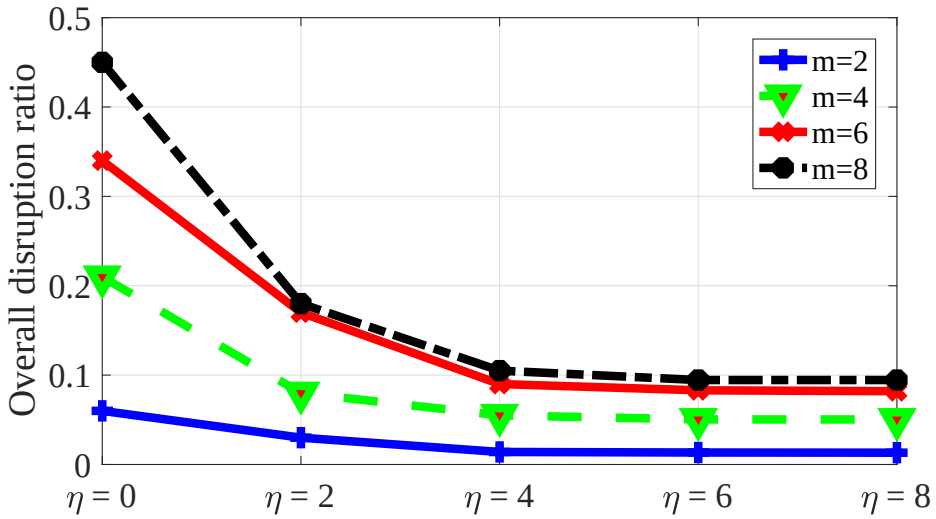


Figure 2.12. The overall disruption ratios given different value of η with different m.

2.5.3.3 Varying the Number of Malicious Nodes

Given the parameters c_1, c_2 and η, we compared the influence-limiting defense with existing defenses in cooperative spectrum sensing.

We first compared the defense performance of four cases in cooperative spectrum sensing: (i) no LEB attack and no influence-limiting defense at the fusion center; (ii) LEB attacks are present without the influence-limiting defense; (iii) No LEB attack while influence-limiting defense is present; (iv) both LEB attacks and influence-limiting defense are present.

We compared the performance in terms of the overall disruption ratio for $m = 2, 4, 6, 8$ in Table 2.5, which demonstrates that the cost of influence-limiting defense is slight when no malicious node is present (with an overall disruption ratio of 0.008 compared to 0.002 when influence-limiting defense is absent). Compared with the case when no influence-limiting defense is applied, the average defense performance improvement of influence-limiting defense in terms of overall disruption ratio is around 78% when LEB attacks exist, which shows the effectiveness of the defense.

Table 2.3. The overall disruption ratios with different numbers of malicious nodes m.

LEB attacks	**Influence-limiting defense**	m			
		2	**4**	**6**	**8**
absent	absent	0.002			
present	absent	0.06	0.21	0.34	0.45
absent	present	0.008			
present	present	0.013	0.05	0.082	0.095

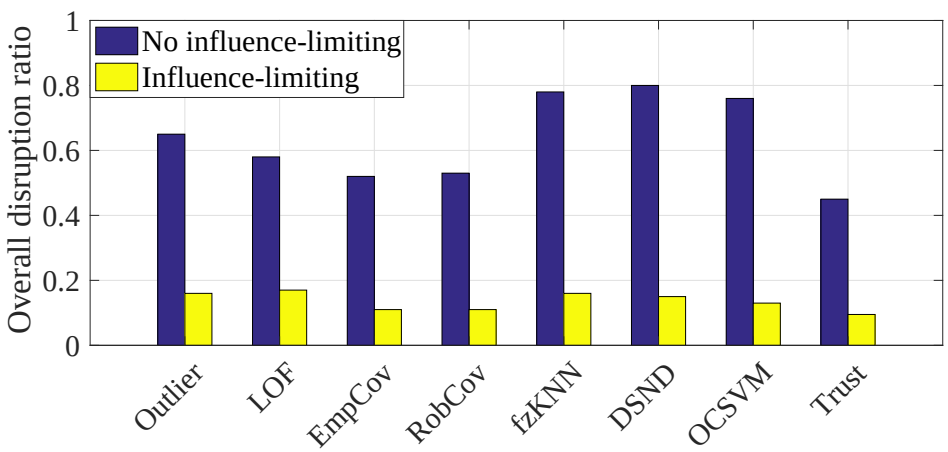


Figure 2.13. The overall disruption ratio comparison of influence-limiting policy for different defenses.

2.5.3.4 Performance Comparison with Existing Defenses

Next, we compare the performance of influence-limiting defense with other existing defenses in terms of overall disruption ratio. We employed the same configurations for parameters of influence-limiting defense and other defenses, and set $m = 8$. The performance illustrated in Figure. 2.14 validates our proposed defense by dramatically decreasing the overall disruption ratio by around 80% on average.

The above defenses are general intrusion detection methods used to detect outliers. We also compared influence-limiting defense with specialized defense methods designed specifically for cooperative spectrum sensing. Double-Sided Neighbor Distance (DSND) [10] is a malicious nodes detection method based on the assumption that *if a sensing node's history*

Table 2.4. Overall disruption ratio comparison under different defense methods.

Defense methods	**Without defense**	**With defense**
Li et al.[10]	0.85	0.36
Chen et al. [17]	0.85	0.33
Qin et al. [81]	0.85	0.31
Hyder et al. [82]	0.85	0.38
Influence-limiting defense	0.85	0.27

is too far away or too close to other's histories, then it might be abnormal and is probably a malicious node. A malicious node identification method and an adaptive linear combination rule were provided in [17]. Besides defenses of IMBA, a modified Combinatorial Optimization Identification (COI) algorithm was proposed in [81] to deal with cooperative attacks. An adaptive reputation-based clustering algorithm was presented in [82] to defend against both IBMA and CMBA.

We quantified defense performance in terms of overall disruption ratio to compare influence-limiting defense with existing defenses in cooperative spectrum sensing. LEB attacks are deployed as the attack method and the malicious nodes are set as $m = 8$. The fusion center is configured with SVM as the decision rule. Unlike previous experiments where we run the influence-limiting defense with other defenses, here we run the influence-limiting defense as the only defense mechanism at the fusion center. The detailed performance is shown in Table 2.4.

The performance of influence-limiting defense is slightly better than existing methods even run independently. LEB attacks have better attack performance on reputation-based defenses like method in [82], which ends up with 38% overall disruption ratio, than other methods.

The experiment results above demonstrate the robust performance of the proposed influence-limiting defense. Influence-limiting defense can defend against both independent Byzantine attacks and cooperative Byzantine attacks. Similar to many security mechanisms, there

will be a performance cost especially when no malicious nodes are present. How to better balance the security and performance is identified as potential future work.

2.5.4 Low-cost Influence-limiting Defense Experimental Analysis

2.5.4.1 Experimental Configurations

To fully compare the performance of the original influence-limiting defense and our low-cost version of influence-limiting defense, we employ the same experimental configuration in [63], in which we have 20 sensing nodes in the cooperative spectrum sensing network, and vary the number of malicious nodes to fully demonstrate the defense performance under attacks of different powers. We also focus on defending against the LEB attack using the same dataset as collected in [63]. In the configuration of the fusion center, we consider eight existing representative intrusion detection defenses and using Support Vector Machine (SVM) algorithm as the fusion rule. The configuration for the LEB attacker is also the same as that in [63]. The performance metric that we use in our comparison is the *overall disruption ratio*, which is defined as the ratio of successful attacks over the total elapsed timeslots (in each timeslot, a sensing node will report a sensing result to the fusion center):

$$\text{Overall disruption ratio} = \frac{\text{number of successful attacks}}{\text{number of elapsed timeslots}}. \tag{2.12}$$

2.5.4.2 Results and Analysis

In our experiments, we first provide the results of our low-cost influence-limiting defense against the LEB attack with different number of nodes controlled. We adopt the same parameter configuration as that in [63], in which the cost control parameter c_1 is set as 0.05 and the influence control parameter c_2 is set as 0.5. We measure the performance of our low-cost influence-limiting defense and compare it to the original version under different number

Table 2.5. Performance comparison under different number of malicious nodes m.

	m				
	0	**2**	**4**	**6**	**8**
Originial version[63]	0.008	0.013	0.050	0.082	0.095
Proposed low-cost version	0.007	0.014	0.043	0.085	0.105

of malicious nodes. The results are shown in Table 2.5, from which we observe that when there is no LEB attack, the performance cost of our low-cost version is slightly better than the original version of influence-limiting defense. However, as the number of malicious nodes increases, our proposed defense version performs slightly worse than the original version. It is reasonable since when there are malicious nodes in different subsets, there is a small chance of not limiting the attack power of those malicious nodes that exist in the subset where they do not reach the threshold defined by the cap function δ.

In the second group of experiments, we evaluate the overall disruption ratio of the low-cost influence-limiting defense together with different existing defense mechanisms in the fusion center. The number of malicious nodes is set as $m = 8$ for the LEB attack. The results are shown in Figure 2.14, from which we observe that the low-cost version performs slightly worse than the original version of the influence-limiting defense. The reason is similar to that observed in the first group of experiments. As we set $m = 8$, some malicious nodes do not get their weights limited by the defense.

In the third group of experiments, we compare the complexity cost of the low-cost version of the influence-limiting defense with the original version. In Table 2.6, we compare the averaged decision time needed when deploying the two versions of influence-limiting defenses. We use a normalized time cost to compare the two methods and set the time cost for the original version of influence-limiting defense as 1 when the number of malicious nodes is 0. The results show that the time cost for our proposed low-cost version of influence-limiting defense is less than 20% of the time cost when running the original version of the defense,

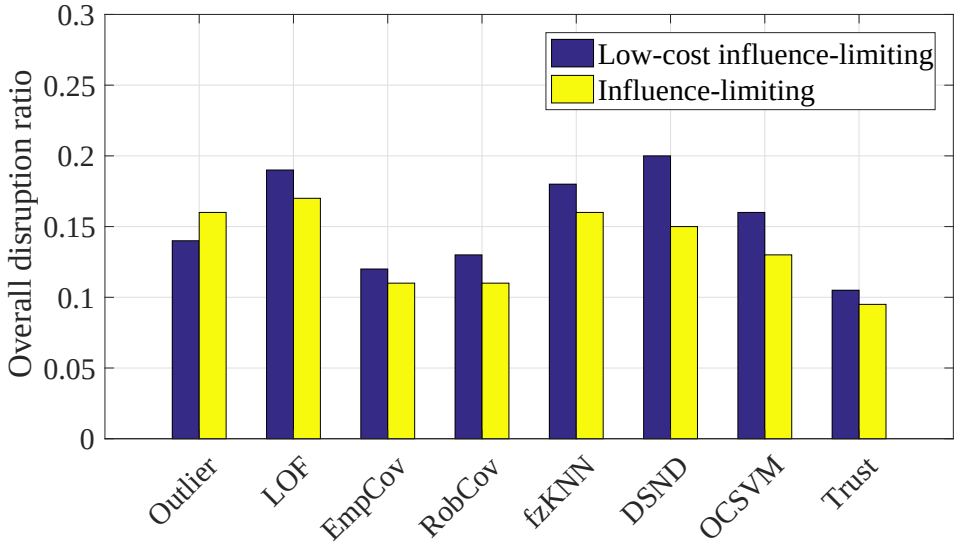


Figure 2.14. The overall disruption ratio comparison.

Table 2.6. The normalized average decision time cost comparison.

m	Original version [63]	Proposed low-cost version
0	1.00	0.10
2	1.06	0.11
4	1.08	0.15
6	1.09	0.16
8	1.11	0.18
10	1.12	0.19

which demonstrates the great advantage regarding low complexity of the proposed low-cost version of influence-limiting defense.

In the fourth group of experiments, we examine the normalized time cost regarding different numbers of malicious nodes. We set the time cost when there exists only one single malicious node as 1. The results are shown in Figure 2.15, from which we can observe that when malicious nodes cover the half of all the sensing nodes, the time cost is only around 20% compared to the scenario when the number of malicious nodes is 1. The reason is that in the low-cost influence-limiting defense, when the number of malicious nodes increases, their attack capabilities accumulate more quickly under the same configuration. Thus, it becomes

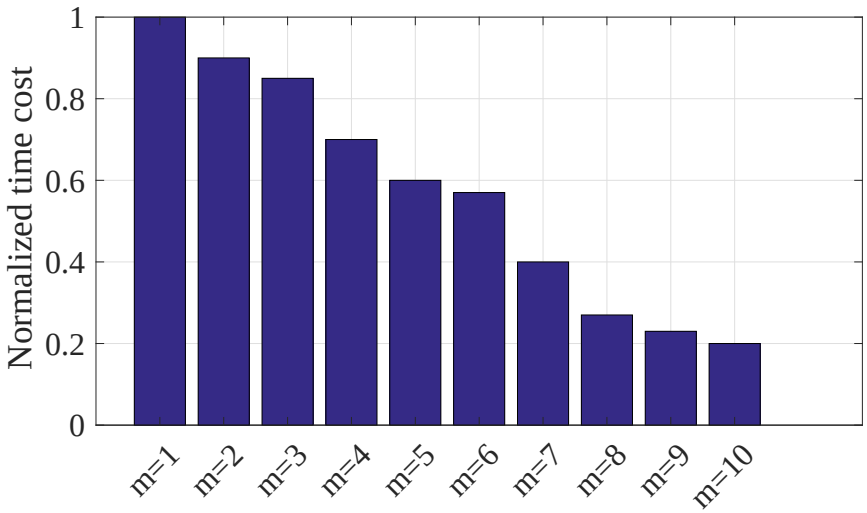


Figure 2.15. The normalized time cost under different number of malicious nodes.

easier to reach the threshold defined by the cap function $\delta(|X_{\text{sub}}|)$. Further, it becomes easier to trigger the influence-limiting defense, enforcing the weight limitation of $\mathcal{X}_{\text{sub}}$. When the number of malicious nodes is small, it is necessary to examine more rounds of "divide" and evaluate the decision-flipping influence of the corresponding subsets, which requires more time.

The proposed low-cost influence-limiting defense has comparable performance and has much lower complexity. It only incurs a very modest defense performance loss while achieving much better run time efficiency. However, there is still a room to improve. For example, is there any better solution to do the "divide"? Can we achieve even better defense performance while decreasing the run time cost significantly? These aspects are identified as future research direction.

2.5.5 Related Work

Cooperative spectrum sensing is an efficient way to detect spectrum usages of TV white space channels. Defenses like statistics-based, machine learning-based, and trust-based meth-

Algorithm 3 Low-cost influence-limiting defense

Input: Historical sensing results and the corresponding decision outputs in $\mathcal{Y}$; parameters c_1, c_2.

Divide $\mathcal{X}$ evenly into two subsets $\mathcal{X}^1_{\text{sub}}, \mathcal{X}^2_{\text{sub}}$;

do :

If any of the two subsets is an empty set:

Compute $I(\mathcal{X}_{\text{sub}})$ of the non-empty subet and enforce influence if it is larger than $\delta(\mathcal{X}_{\text{sub}})$; otherwise terminate the defense process.

else:

Compute $I(\mathcal{X}^1_{\text{sub}}), I(\mathcal{X}^2_{\text{sub}})$;

If $I(\mathcal{X}^1_{\text{sub}}) < \delta(|\mathcal{X}^1_{\text{sub}}|)$ and $I(\mathcal{X}^2_{\text{sub}}) < \delta(|\mathcal{X}^2_{\text{sub}}|)$:

terminate, no malicious subset is detected;

If $I(\mathcal{X}^1_{\text{sub}}) \geq \delta(|\mathcal{X}^1_{\text{sub}}|)$ and $I(\mathcal{X}^2_{\text{sub}}) < \delta(|\mathcal{X}^2_{\text{sub}}|)$:

Update the two subsets $\mathcal{X}^1_{\text{sub}}, \mathcal{X}^2_{\text{sub}}$ with two new evenly divided subsets from $I(\mathcal{X}^1_{\text{sub}})$;

If $I(\mathcal{X}^1_{\text{sub}}) < \delta(|\mathcal{X}^1_{\text{sub}}|)$ and $I(\mathcal{X}^2_{\text{sub}}) \geq \delta(|\mathcal{X}^2_{\text{sub}}|)$:

Update the two subsets $\mathcal{X}^1_{\text{sub}}, \mathcal{X}^2_{\text{sub}}$ with two new evenly divided subsets from $I(\mathcal{X}^2_{\text{sub}})$;

If $I(\mathcal{X}^1_{\text{sub}}) \geq \delta(|\mathcal{X}^1_{\text{sub}}|)$ and $I(\mathcal{X}^2_{\text{sub}}) \geq \delta(|\mathcal{X}^2_{\text{sub}}|)$:

Enforce influence through limiting the weight $w(\mathcal{X}^1_{\text{sub}}), w(\mathcal{X}^2_{\text{sub}})$;

until $\mathcal{X}^1_{\text{sub}}, \mathcal{X}^2_{\text{sub}}$ are both are empty sets:

ods have been widely studied. Statistics-based defenses assign different statistics to sensing nodes. For example, an outlier is computed for each node in [12]; [17] obtains weighted coefficients of each node; and [18] proposes a method through majority vote of neighboring nodes, a Bayesian method is discussed in [19]. Machine learning-based methods leverage machine learning techniques to classify legitimate and malicious data. For example, [14, 22] utilize supervised learning based on SVM to distinguish malicious nodes; [10] uses different KNN-based algorithms in an unsupervised way to detect malicious nodes; and [20, 21] discuss both supervised and unsupervised ways to defend the fusion center. A trust-based method [11, 13, 58] maintains a trust value for each node, which will be used as a weight in the global decision process.

Taking into account such a wide range of defenses, we present a powerful attack mechanism based on the LEB attack framework, which aims to learn the defense at the fusion center and then launch effective attacks. Based on the proposed attack strategy, we introduce an influence-limiting defense, which is a non-invasive method that can coexist with existing defenses to bridge the gap between traditional defenses and new LEB attacks.

There are some existing studies that deal with smart/intelligent attackers in cooperative spectrum sensing. In [83], a smart attack framework that can maximize the expected aggregated reward of the attacker was proposed. In [83, 69], smart defense mechanisms were proposed to discourage attackers. Our work differs from these studies in that the LEB attack framework focuses on using a black-box strategy to minimize the attacker's trace (via LEB) and the malicious perturbation when the attacker decides to launch the attack.

Adversarial machine learning focuses on learning under the existence of active adversaries [84]. The transferability [24] in machine learning models gives adversaries the opportunity to learn and compromise targeted models. Adversarial example generation methods, such as iterative methods [71, 70] and gradient-based methods [54], are well-known ways to create malicious data targeting a machine learning classifier under specific scenarios. Our proposed adversarial sensing data generation algorithm achieves similar performance as other existing methods, while reducing by 65% the computational cost on average.

There is a growing interest in applying adversarial machine learning to wireless communications [85]. Exploratory (inference) attacks were studied in [86, 87] to build a surrogate model (to mimic wireless transmission patterns) that is used to develop a smart jamming scheme. Causative (poisoning) attacks were developed to manipulate the training data input to machine learning classifiers [88, 89, 90] with applications in wireless access and IoT. A similar approach was followed in [91] to design a Trojan attack, where the adversary adds Trojans (triggers) to the training data and activate them later in test time against a wireless signal classifier. On the other hand, membership inference attacks were considered in [92]

to learn whether a particular data sample has been used to train a wireless signal classifier. Finally, evasion attacks were studied in [93], where an adversary jams the sensing period to change the inputs to a machine learning classifier and force a target transmitter into making wrong transmit decisions. Evasion attacks were also considered to fool modulation classifiers [94, 95, 96, 97, 98, 99, 100]. Compared to previous works, this work considers adversarial machine learning attacks against cooperative spectrum sensing in wireless communications and presents corresponding defenses.

2.5.6 Limitations and Discussion

Although our proposed LEB attack framework and influence-limiting defense achieved promising attack and defense performance, it is important to note the cost of the fusion center. We conducted experiments from different aspects to validate the LEB attack framework. However, it can not guarantee attack performance. The LEB attack framework is based on machine learning techniques, which still have many unanswered questions [23, 55, 51]. We list limitations of the LEB attack framework as follows:

- How to choose the minimum number of and optimal sub-models in the surrogate model depends on specific scenarios. In the design, we make the sub-model as a flexible algorithm set such that new models can be added and current sub-models can be removed. But how to decide the number and types of sub-models in the surrogate model in practice is left as a design choice, which we identify as potential future work.

- We need to train and update all the sub-models concurrently in the training and testing process. In our experiments, the average convergence time for most sub-models is approximately 500 timeslots. Although we have employed incremental/online learning techniques to minimize the update cost, a better solution still lies ahead as potential future exploration.

- Experimental results of the LEB attack framework are data-dependent. i.e., the attack performance is empirical and might change for different datasets. How to prove the effectiveness of the LEB attack framework from a theoretical perspective is also identified as potential future work.

From a broader AI perspective, our proposed LEB attack framework can be applied to many other scenarios that require data fusion process such as in Internet of Things (IoT) [101]. Another important application is in machine learning itself. Machine learning-based attack framework does not necessarily provide explainability. For example, machine learning models like deep neural networks are treated more like a black box without giving much transparency and accountability to the users. Machine learning models are usually data-driven [23, 55], thus the performance is usually empirical and highly dependent on the given datasets. In our proposed LEB attack framework, we employ an algorithm set to counter the uncertainty such that the attack utility has higher probability to achieve its goal. Besides, we employ the algorithm set to mutually validate the malicious inputs (malicious samples will be taken as inputs for each sub-model).

From the defense perspective, the influence-limiting defense is effective in limiting the damage caused by attacks. It can be also applied to a broader machine learning scenario. However, as shown in experimental results, it will also lead to performance cost when there is no attack. Besides, to enforce a full version of the influence-limiting defense, the computation cost still needs to be addressed. In this work, we explored the defense from a direct way, which is to limit the influence of sensing nodes. Our design of $\delta(|\mathcal{X}_{sub}|)$ provides a feasible and effective trade off between performance and complexity, and we found that in practice it is not necessary to enforce a full-version of the influence-limiting defence to achieve a satisfactory performance.

As machine learning provides IoT systems with powerful means of learning from data and solving complex tasks, it also raises security concerns due to its vulnerability to adversarial

manipulation. Our proposed adversarial machine learning based partial-model attack model focuses on the IoT data fusion process and equips the adversary with the capability to launch successful attacks even when the adversary controls a small part of the IoT devices by exploiting the performance uncertainty of the IoT devices or the communication channel. How to counter the proposed attack is our future work. Below, we provide several potential mechanisms for defense: *Deploying robust anomaly detection mechanism in the IoT fusion center*. This is a direct method to defend the IoT systems against the partial-model attack. However, in this attack, all the manipulated devices cooperate with each other to launch the attack. Thus, how to design an anomaly detection method to detect a set of devices is a challenge. *Improving privacy protection in every level of the IoT infrastructure*. In the partial-model attack, the key to learn a partial model to mimic the fusion center is the availability of the output of the fusion center. Thus, the decision information can be kept as private and secure by deploying a privacy protection mechanism.

Using machine learning to attack the IoT systems is detrimental to the IoT security. On the other hand, machine learning can be also employed as a defense method [102]. Therefore, it is important to understand the interaction of machine learning techniques used for attack and defense, and game theory can be used as mathematical means to study the conflict of interest driven by machine learning.

Chapter 3: Security of HPC Systems and Software Systems

In this chapter, we explored the security attacks and defenses in HPC systems and software systems. Especially explored how the machine learning techniques could be deployed to detect the anomalies or malicious programs. This chapter was published in EAI Endorsed Transactions on Security and Safety, IEEE INFOCOM 2020-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS). Permissions are included in Apendix A.

3.1 Background and Related Work

In this section, we give the detailed background of log-based anomaly/intrusion detection in HPC systems and the mathematical model of reinforcement learning, which is the foundation of our ReLog framework.

3.1.1 Machine Learning and HPC Security

Besides the aforementioned intrusion detection methods, there proposed various advanced frameworks in recent years to mitigate security risks, especially as machine learning techniques are widely used in HPC systems.

For data centers like NERSC, there are hundreds, or thousands of jobs running concurrently on each of their four computing systems, i.e., Cori, Edison, PDSF and Genepool every day. Inspecting and analyzing each job manually is practically infeasible given the vast amount of log files generated simultaneously by each of the systems and the corresponding file systems. Given the huge volumes of log files, a mechanism named as Priolog is designed

in [103] to narrow down the volume to comparatively small volumes of most related logs, thus increases the process efficiency.

Besides reducing the volumes of logs, another idea is to design scalable HPC system data analytics framework. [34] designed a scalable mechanism to analyze system logs, further, to distinguish unwanted or malicious jobs running on HPC systems. Based on the framework, users are able to navigate spatial-temporal event space that overlaps with specific system resource allocation, events, errors and identify persistent user behavior patterns etc. Further, users can distinguish performance anomalies and gain valuable insights about the impact brought up by various system jobs. Ultimately all above methods lead to machine learning techniques. Given the often-huge amount of log files generated during the running stage (we say it "huge" because log files are usually generated continuously from various layers of the systems) [104], traditional data analysis tools and techniques often fall short of the capability of processing them efficiently and effectively. Machine learning techniques, such as deep neural networks, can offer exactly what traditional data analysis techniques fall short of. It can process large dataset of log files in a batch-processing style and can extract valuable information from them.

In log analysis, machine learning techniques have their unique advantages in dealing with large volumes of streaming data. Processing streaming data generated from HPC systems is challenging as a result of the large volumes and generating speed. An online supervised learning method is proposed in [105] to operate with live streamed data; Another online anomaly detection method using autoencoder is provided in [106]. To better support streaming logs analysis, a visual analytic framework is proposed in [107], which consists of data management, analysis and interactive visualization. It can automatically identify pattern changes and provide a coherent view of the changes and patterns of the performance data. [108] also builds a scalable visualization tool named as MELA to study event log data.

Other applications of machine learning in log file analysis can be found in [109, 110, 111, 112, 113]. The application of machine learning techniques in log file analysis brings revolutionary to the intrusion/anomaly detection in HPC systems. Anomaly user detection is an important application of deep learning techniques for user behavior analysis. A comprehensive review of how the machine learning techniques are used in detecting anomaly users is given in [5].

3.1.2 Reinforcement Learning

Reinforcement learning process is usually modeled as a Markov decision process (MDP) with a state space $\mathcal{S}$, an action space $\mathcal{A}$ and the Markov property in terms of stationary transition dynamics as $p(s_{t+1}|s_1, a_1, ..., s_t, a_t) = p(s_{t+1}|s_t, a_t)$ of any sample trajectory $s_1, a_1, s_2, a_2, ..., s_T, a_T,\ s \in \mathcal{S}, a \in \mathcal{A}$ with initial state distribution $p(s_1)$. The reward/cost function is termed as $\mathcal{S} \times \mathcal{A} \to \mathbb{R}$. The reward $r_t^\gamma = \sum_{k=t}^{\infty} \gamma^{k-t} r(s_k, a_k)$ is the total discounted reward starting from time t, $0 < \gamma < 1$ is the discount factor.

The rule of selecting an action given a state in MDP is named as policy functions, which have the form of $\pi_\theta : \mathcal{S} \to P(\mathcal{A})$, $P(\mathcal{A})$ is the probability distribution over the actions. $\theta \in \mathbb{R}^n$ is a vector of parameters of the policy function. Value functions are the expected total discounted reward. For a given state s, the value $V^\pi(s) = \mathbb{E}[r_1^\gamma | s; \pi_\theta]$ and for a given action a over state s, the value $Q^\pi(s, a) = \mathbb{E}[r_1^\gamma | s; a; \pi_\theta]$. Therefore, we can write the general objective of the reinforcement learning as an expectation $J(\pi_\theta) = \mathbb{E}_{a_k \sim \pi_\theta}[r(s, a)]$ [114].

There are three main different perspectives to solve $J(\pi_\theta)$:

- *Value function algorithms* [115] update value functions after each sample trajectory. The optimal action at each state is the action that can achieve optimal value functions V_* or $Q_*^{\pi_\theta}$ according to Bellman optimality equations, in which $V_*(s) = \max_a \mathbb{E}[R_{t+1} + \gamma V_*(S_{t+1}) | S_t = s, A_t = a]$ and $Q_*^{\pi_\theta}(s, a) = \mathbb{E}[R_{t+1} + \gamma \max_{a'} Q_*^{\pi_\theta}(S_{t+1}, a') | S_t = s, A_t = a]$.

R_{t+1} is the transition reward from state $S_t = s$ to S_{t+1}. $S_t, S_{t+1} \in \mathcal{S}, A_t \in \mathcal{A}$. a' is the action at state S_{t+1}.

- *Policy gradients* [116] update policy parameter vector θ directly through $\theta \leftarrow \theta + \alpha\nabla_\theta J(\theta)$, in which $\nabla_\theta J(\theta) \approx \sum_i(\sum_t \nabla_\theta \log\pi_\theta(a_t^i|s_t^i))(\sum_t r(s_t^i, a_t^i))$ if the commonly known REINFORCE algorithm is used. α is an adjustable parameter to control the step size. a_t^i, s_t^i denote the action and state at time t from sample trajectory $\{\tau^i\}$.

- *Actor-critic algorithms* [117] combine both the value functions and policy gradient update methods. Using value functions, e.g., advantage functions, to estimate $\sum_t r(s_t^i, a_t^i)$ of $\nabla_\theta J(\theta)$ and following the same update strategy in (ii).

We model our anomaly/intrusion detection in HPC as a sequential decision problem. Each feature vector is modeled as a state, and we employ value function algorithms to update the transition reward. Based on the cumulative reward metric, we make the decision of whether the evaluated user is anomalous or not.

3.1.3 Attacks in HPC Systems

Attacks in HPC systems can lead to unauthorized accesses to the high-performance system infrastructure with malicious intents to steal, compromise or vandalize HPC resources, e.g., hackers might vandalize the system on purpose if they failed to extract any useful information or exploit the system to launch a denial-of-service attack [118].

The security in HPC facilities is similar with those in typical IT context. They are connected to networks just like other computer systems, and often run on the Linux-based systems. Thus, many of the attack styles in typical IT context can also be applied in HPC facilities [119]. However, HPC systems also have their own vulnerabilities, because they usually run exotic hardware and software systems, and have highly different purposes and modes of use compared with typical IT systems.

Probes, scans, brute-force login attempts, and buffer overflow vulnerabilities are the common issues that trouble HPC facilities. Common vulnerabilities of HPC facilities can be listed as follows[26]:

- Intrusion attacks, which can lead to data leakage or other consequences. It is because HPC facilities are extremely "open" to users around the world, thus make them an easy target for attackers.
- Alteration of code or data.
- Misuse of computing cycles, i.e., using the HPC facilities to perform activities other than granted behaviors such as crypto-currency mining.
- Availability related threats, including disruption or denial-of-service attacks against HPC facilities.

Threats confronted by HPC systems can be classified into three well-known types: confidentiality (e.g., data leakages), integrity (e.g., alteration of data or code) and availability (e.g., disruption/denial-of-service attacks against HPC systems or networks that connect them) [120]. The cause of these attacks can be insiders or outsiders of the HPC system. There are two kinds of attacks that are most favored by the outsiders: the brute-force attack against the password system and man-in-the-middle attacks [121].

"Insiders" often have special advantages in terms of HPC security because they can access HPC infrastructure. Thus, it is much easier for insiders to launch attacks than others [119, 122], e.g., malicious users can change or modify the data stored in the system, or it can manipulate the system to do some illegal things such as spreading the rumors or viruses on the Internet. Even more, they can manipulate the system to run harmful programs to consuming the system resources.

Given the vulnerabilities and characteristics of HPC facilities, we list four representative attacks in Linux cluster-based HPC facilities, which can be launched against an HPC cluster architecture:

3.1.3.1 Daemon Process-based Attack

A daemon process is an application that does not need interaction with a terminal and running in background. It can be launched by any other process, and it can be applied to take over system resources even when the parent process has terminated execution. This sort of attack can be very powerful in MPI environment since when a daemon process is created, the MPI scheduler will lose track of it [118]. The daemon process can take over any kind of CPU usage or other computing resources.

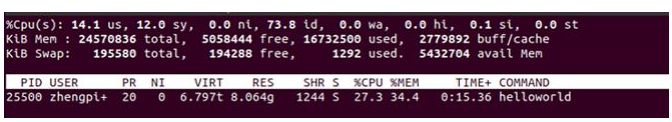
To emulate this type of attacks, one need to insert a Trojan into a trusted application appropriately to avoid being found by the HPC administrator. Example of Daemon process-based attacks include daemon memory allocation attack (running a daemon process to take over the memory resources) and daemon file attack (running a daemon process to take over the storage, memory, and computational resources of HPC facilities). An example of the daemon process-based attack template is shown as in Listing 3.1:

Listing 3.1. Daemon process-based attack example [118].

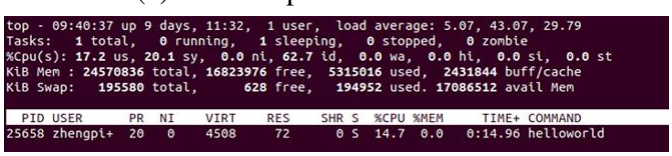
```
void main(){
int pid;
void *pointerMemory;

pid = fork();
if(pid < 0)
exit(EXIT_FAILURE);
if(pid > 0)
exit(EXIT_SUCCESS);
setsid();
signal(SIGHUP,SIG_IGN);
umask(0);
chdir("/");
pid = fork();
if(pid < 0)
exit(EXIT_FAILURE);
```

```
if(pid > 0)
exit(EXIT_SUCCESS);
signal(SIGPIPE,SIG_IGN);
openlog("helloworld daemon",LOG_PID,LOG_DAEMON);
int i = 1;
while(i < 100000000){
/*the attack method is put here*/
pointerMemory = (char*)malloc(SIZE);
syslog(LOG_NOTICE,"memory is allocated!");
i++;
sleep(DURATION_SEC);
}
syslog(LOG_NOTICE, "Terminated!");
closelog();
}
```

We run the code on Linux systems, the comparison of CPU and memory utilization between running the attack method in the example and without running the attack is shown in Figure. 3.10, from which we can observe that when the system keeps allocating memory pointer, the CPU time increased from 14.1 μs to 17.2 μs, while CPU time proportion for the user increased from 14.7% to 27.3% and memory usage increased from 0.0% to 34.4%. It demonstrates the attack power of a daemon process-based attack.



(a) Daemon process based attack



(b) Normal daemon process

Figure 3.1. Resource usage of daemon-process attacks.

3.1.3.2 Interposition Library Attack

It is an attack by intercepting function calls that an application makes to the stack of share libraries, and then modify the called function to add malicious functionality to it. In Linux systems, interposition is ”the process of placing a new or different library function

between the application and its reference to a library function" [123]. Example interposition library attacks include attacking the libc library and attacking the MPI library. An example of the interposition library attack is shown in listing 3.2.

Listing 3.2. Interposition library-based attack example [118].

```
static int DoProfile = TRUE;
FILE *fopen(const char *filename,const char *mode){
typedef FILE*(*function_type)(const char *filename, const char *mode);
static function_type function = NULL;
static char *function_name = "fopen";
FILE *retval;
if(!function){
function = (function_type)dlsym(RTLD_NEXT, function_name);
}
if(DoProfile){
DoProfile = FALSE;
retval = (*function)(filename,mode);
DoProfile = TRUE;
}
else
retval = ((*function)(filename,mode));
sys_call_table[SYS_open] = orig_open;
}
```

3.1.3.3 Probe-based Login Attack

It is similar with brute-force login attack[124, 125]. When the username of the HPC facilities is known to the attacker, they can create a dictionary of all potential passwords to launch the password guessing campaign as HPC facilities usually are "open" to users all around the network.

3.1.3.4 Intrusion Attack

Given all the security issues in HPC systems, we found that intrusion attack is probably one of the most discussed attacks[31, 35, 7, 126, 4]. Because for any malicious user to launch an effective attack, the primary obstacle is how to access the targeted system. Thus, many problems of security for HPC systems are centered around intrusion attacks and detection.

Some commonly known intrusion attacks include attempts to copy the password file frequently, a lot of unreliable remote procedure call requests in a very short time and attempts to connect to non-exist "bait" hosts frequently [126].

Besides attack examples listed above, there are many other attack methods in HPC systems [119, 26, 4]. Based on all these various attacks, how to guard the HPC system against these attacks becomes a challenging task. To the best of our knowledge, different defenses have been proposed to defend against a set of attacks, while there are no universal solutions.

When a server or cluster comes under attacks or once the attacker lands on the system, owners of user accounts often have no idea of the reality that they have been hacked. Most of the time, it is the activities of attackers expose or alert security officers of the HPC infrastructure. Behaviors of hackers tend to fall into some modes that are obviously different from the true owner of the account and can be easily identified. For example, the sudden burst of network activities, high CPU utilization, longer network latency or unauthorized jobs bypassing the job scheduler etc. Hence monitoring is an important part of the HPC system management, which is a main task for many defense mechanisms.

3.1.4 Defense Mechanisms in HPC Systems

In this section, we give the background and defense strategies in HPC systems. We especially reviewed intrusion detection methods due to the importance in HPC security.

Early studies on the security of HPC systems focused on the programming level. A software infrastructure is designed in [27] to ensure the integrity and confidentiality of communications and to authenticate approved users and resource owners. The developed security-enhanced communication library, named as Nexus, can be used to provide secure versions of popular communication libraries, such as Message Passing Interface (MPI), and offered a

fine degree of control over what, when and where security mechanisms can be employed in HPC systems.

Traditionally, many physical measures and human-oriented rules have been suggested to ensure the security of HPC systems. For example, Korambath et al. in [119] listed strategies of protecting passwords and how to safeguard computing resources in an HPC environment. Common methods of preventing passwords from hacking include encrypting information exchange between users and resource owners, urging users to employ complicated passwords and changing the passwords routinely, monitoring activities of each user to identify possible attackers and limiting the access rights of each user according to their priority and so on.

Protocols like telnet and ftp are widely used in the 1990s in which clear text format information is transferred between remote computers. Anybody with reasonable expertise of the domain knowledge can launch such an attack, intercepting and reading the content. Now none of HPC sites will run these kinds of protocols. Besides that, a lot of solutions to malicious user detection and security risk reduction have been proposed. The approach used by most of nowadays HPC systems can be summarized as divide-and-conquer. i.e., the security techniques are deployed separately and independently to address the HPC security vulnerabilities [119].

What to look for and where to identify the malicious attacker are a complicated cyber-security problem. Not a single HPC suit, especially within the field of open science, has any solution to identifying and reducing all risks associated with attacks against the system infrastructure. Given the sophistication of attackers and the rapidly changing pace in computational systems and networking, how to develop defense mechanisms becomes a difficult obstacle to tackle with.

When HPC systems are under attacks, the attack activities often have two parts: the initial attack and the follow-up behaviors taken if local access is obtained. The initial attack can come from almost anytime and anyplace. Although we have firewalls in place, there is

nothing the firewall can do once the attacker has already landed on the system. However, as of the attack property, the attacker's behavior tends to be somehow well-defined and can be identified easily. From the system administrators' perspective, what to look for and where to identify the malicious attackers are probably one of the most complex cybersecurity problems in nowadays HPC environment [31].

One the other hand, security issues in HPC systems differ from those in traditional platforms due to their distinctive program structures, computing environments and performance requirements. Unlike widely used client-server structure in traditional distributed systems, the communications in HPC systems mainly depend on two-sided message passing, streaming protocols, multicast and so on [27].

Existing technologies used to secure HPC infrastructures can be broadly categorized into four classes, i.e., they try to defend HPC systems from the following four aspects[3]: (i) basic OS hardening, (ii) authentication, (iii) network and host security and (iv) patching and auditing, in which OS hardening techniques can go deeper to be classified into different sub-classes. The detailed category information is shown in Figure.3.2. From which, password complexity enforcement can be done through enforcing strict well-defined security policies, for example, requiring users to change password periodically; Monitoring HPC facilities for abnormal status [5] can be one example of defenses under authentication category; As HPC facilities are usually used for distinctive purposes such as mathematical computations, they tend to have a much more regular and predictable mode of operations, which can be utilized to detect irregular and potential malicious attacks [4].

Intrusion detection is a well-known defense in HPC systems to counter malicious attacks [35]. To address the problem of where, when, what and how to identify the suspicious intruders, National Energy Research Scientific Computing Center (NERSC) [31] follows a methodology of data gathering and measurements, repeatable testing and careful analysis

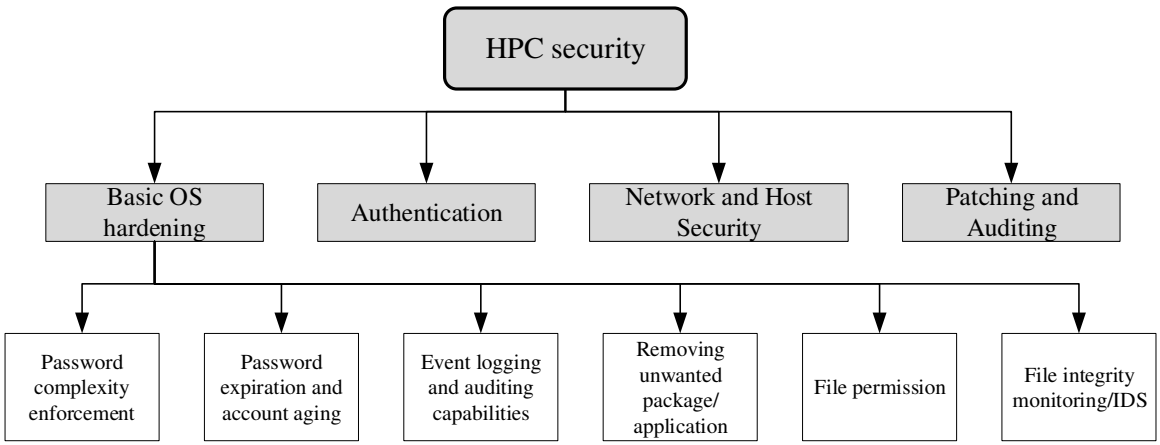


Figure 3.2. Strategies to secure HPC systems [3].

in designing the intrusion detection mechanism. The detailed steps of the method can be summarized as follows:

- Collecting as much raw data as possible, especially those data in high yield areas, and designing patterns that can be based on to evaluate the suspiciousness of user behaviors. The raw data can be accessed through accounting data or SSHD data from each host, batch scheduler logs from inter-systems or network data and DNS logs from cross-site.

- Cleaning, organizing and normalizing the collected data for the following analysis, which aim to process and reduce the volume of the collected data. The abstracted data can be further canonicalized to transform it into a normal form such that it can be suitable for machine processing use techniques such as machine learning algorithms.

- Employing appropriate tools or methods to analyze data processed in step 2 and comparing with expected or normal data. In this step, local site security policies can be used to evaluate the standardized data.

Based on the above methodology, NERSC introduced the Bro intrusion detection system [31]. To address the invisibility problem of activities happened on the multi-user HPC

infrastructure, NERSC introduced an instrumentation layer into the OpenSSH application and then connected the resulted dataset into a real time analysis using Bro IDS.

To address various security issues at different layers of the cluster architecture, from threats of network to vulnerabilities of applications, another intrusion detection mechanism is proposed in [33]. In the mechanism, there exists an instrumented node to help gathering raw data and understanding the communications between clustering nodes in HPC infrastructure at various layers (i.e., applications layer, middleware layer, operating system layer and network layer). The collected data will be audited and analyzed to identify suspicious communications or behaviors.

Unlike the intrusion detection techniques listed above, which focus on detecting anomalies from behaviors or running statistics after the intruder has succeed partially in compromising the targeted system. User authentication is another effective method to prevent intruders or malicious attackers from landing the system at the beginning, which is an intuitive and straightforward method to stop intruders. However, this method usually has limited capabilities in keeping safe of HPC systems as many attackers can find paths to walk around the authentication gate [127, 128]. More existing literature of user authentication techniques and mechanisms used in distributed HPC systems can be found in [129, 28, 122].

As machine learning has been widely spread into different domains, a more direct and effective strategy has come into people's mind to defend HPC systems, log file-based defense [109, 130, 4], which is a behavior detection method based on log files generated from running jobs. Using log files has many advantages than previous methods, as log files are a detailed record of the running process of the applications. Log files generated at different layers of the HPC systems often contains the fully operation and resource balance process. We give more information on log file-based intrusion detection in the following section.

3.2 Log-based Security Analysis in HPC Systems

Log files can be found on almost all computer systems universally, which are text files recording behaviors, system status and other running statistics of the jobs active in HPC systems such that administrators can look back to debug system problems or track the running process to find vulnerabilities [131, 132]. Therefore, it can be used as the first and direct information source to monitor and detect intruders in HPC systems [133, 4].

Based on the collected information, the intrusiveness can be downgraded at each level of HPC systems and can even stop monitoring for the sake of efficiency and performance. For example, we can employ two phases in the instrumentation [133, 4]: fully monitoring and adaptive monitoring. Fully monitoring is usually employed at the beginning, in which everything is monitored to build the profile of an application or a user. It is costly in terms of computing resources; thus, it usually only runs a short of time. While adaptive monitoring can adaptively monitor the resources or variables according to the load of the instrumentation node. It can focus on some critical areas while have little impact on the overall system performance.

HPC systems usually can be decoupled into 4 layers [32, 33]: application layer, middleware layer (MPI and PVM, etc.), operating system layer and network layer. The high performance is achieved through the cooperation of each layer. Many of HPC systems are based on Linux-like systems. In Linux systems, user behaviors and operating system's activities are logged such that it can be processed and analyzed by administrators to monitor the system.

When the related raw data are collected by the dedicated instrumented node, all layers of HPC systems are instrumented heavily to evaluate the possibility of existence of intruders[8]:

- *Application layer:* Collect general variables of monitored processes, e.g., percentage of CPU time, memory used, I/O time and so on, to analyze statistics of applications run on HPC systems.

- *Middleware layer:* Audit the calls of messaging passing information, such as MPI calling statements and other middleware related communications.

- *Operating system and network layer:* Audit and monitor access to the file system and related network interfaces, such that the global communications or the local resource access information can be utilized and evaluated.

The general workflow of using log files to detect intruders in existing literature can be summarized and shown in Figure.3.3. Usually we have four steps: log collection, log parsing, filtering/feature extraction and anomaly/intrusion detection, to conduct log file-based analysis in defending HPC systems against intrusions or other malicious behaviors [4, 5, 6, 7, 8].

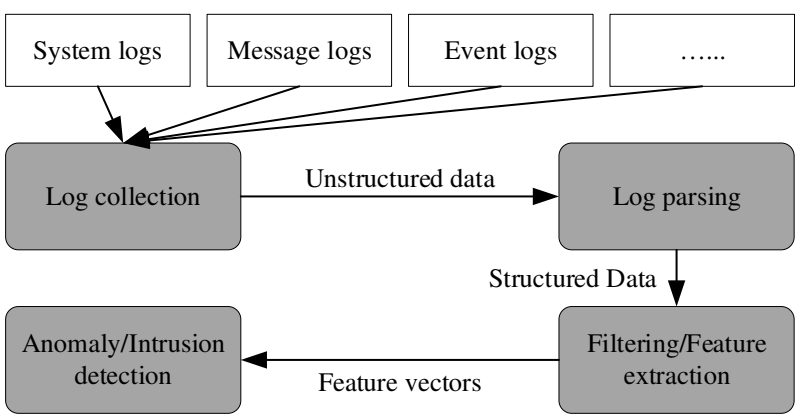


Figure 3.3. Workflow of using logs to detect intruders [4, 5, 6, 7, 8].

3.2.1 Log Collection

Logs record the history of everything happened in HPC systems. In Linux systems, log files are stored in text form and usually under the directory of /var/log and its subdirectories. They include various information such as system, kernel, access control, package managers and many others. These log files can be roughly classified into four categories: application logs, event logs, service logs and system logs.

Large scale HPC systems generate various types of log files during running procedure of the jobs. For instance, the history of application run on the platform, resources allocated for them, sizes of each job, user information of each application and exit statuses are all logged in log files. Reliability, availability and serviceability (RAS) system logs are capable of extracting and logging data from various sensors both hardware and software, such as processor utilization, temperature sensors and memory errors [34]. Other log files like network system logs, input/output and storage system procedure logs can collect and record network bandwidth, congestion, and resource consuming information of jobs. The performance and running information inspection can rely on different log file (s) with respect to requirements of monitoring.

These log files provide a direct way for system managers to evaluate the system status. In HPC systems, log files can also be collected and analyzed to detect malicious jobs or users. For example, message log files in Linux systems show general messages and information regarding the system. It logs all activities throughout the global system. Secure logs keep authentication information of both successful and failed logins, and the authentication processes. Other logs can also be used to detect attacks and evaluate the system vulnerability.

Log collection is the first step in log analysis-based intrusion or anomaly detection of HPC systems. Distributed systems can continuously generate all kinds of log files to record

system states and runtime statistics. This information is vital in examining system condition or debug especially when failures are encountered [134]. In log collection stage, it is important to collect right logs from the system as much as possible to pave way for the following analysis. When machine learning techniques are used to analyze the logs, the volume of the data is critical in learning the patterns of both normal behaviors and abnormal behaviors.

3.2.2 Log Parsing

Log files are usually unstructured data, and their format and semantics might be different from each other, Thus, it is a difficult and complicated problem to design a mechanism that can diagnose abnormal or malicious intruders, especially when log files generated is in huge amount [135, 136, 137]. On the other hand, each log file might contain a lot of information that is not related to the security of HPC systems; thus how to filter valuable information from raw log files is also a challenging problem. It usually requires a lot of domain knowledge to design rule-based detectors [138, 139, 140]. For example, using the CPU time as a resource utilization measure, using IP address to parse a log into different entities and so on.

Log parsing is the second step of log analysis, which aims to get the unstructured, free-format and semantics text file into a structured representation. There have a substantial research and literature on the parsing of log files, in which representative methods can be listed as follows:

- Du et al. [141] provided an *online streaming method* to parse log files, which utilizes the longest common subsequence method, i.e., the longest common subsequence in logs are used as the sign to parse log files into structured files. The proposed method achieves linear time complexity for each log entry.

- Beschastnikh et al. [29] gave a method using *regular expressions* to determine which log lines will be parsed and which log lines will be ignored. It is a set of channel

definitions that corresponds each line of the log files into a vector timestamp, or other channels.

- Xu et al. [142] offered a method leveraging the *source code* to parse log files, in which we need to first get all possible log message template strings from the source code and then match it to log files to parse them into structured files.

- Methods in [143, 144] provided solutions to parse log files purely based on *log characteristics* using data mining approaches. They usually do not rely on other information except log files, which is usually more straightforward but might not be as precise as those rule-based log parse methods, and they usually require machine learning algorithms to learn patterns.

After parsing log files obtained from HPC systems, intrusion/anomaly detection methods can be applied to distinguish out suspicious activities or users.

3.2.3 Log-based Intrusion/Anomaly Detection

There are various methods and mechanisms have been proposed to address the intrusion/anomaly detection problem utilizing log files form HPC systems. In this subsection, we first overview two representative types of log file-based defenses used in HPC systems. They both offered an experiment-proved mechanism to detect intrusions or anomalies:

The first perspective is to take log files as a *language model*, and then build relationships between log files and normal/abnormal behaviors. Log files, in essence, are existed in text format. Therefore, it can intuitively be modeled as a natural language sequence model, further it can employ techniques from natural language processing to analyze log files.

Inspired by the observation that log file entries are a sequence of events extracted from the structed source code execution process, Du et al. [4] proposed a deep neural network model employing Long Short-Term Memory (LSTM) to detect anomaly behaviors and potential

malicious users through taking log files as structured language sequence. Based on the proposed model, which is named as DeepLog, log patterns of normal execution can be learned continuously from the historical data. Abnormal patterns resulted from intrusions, which are often deviates from patterns of log files from normal execution, can be distinguished by the learned model in monitoring process. DeepLog was designed in an incremental learning style, thus it can adapt to different patterns in the running process.

From a mathematical perspective, DeepLog builds a non-linear and high dimensional relationship between log entries and normal/abnormal execution applications. It categorizes log files into various sequences, thus a workflow model can be constructed for each separate task, and also offers a feedback mechanism, such that a wrongly classified log file can be used to adjust weights of the trained model to make it adaptively fit into its dynamic changing environment.

The second perspective is to build *relationships between log files and source codes*. Many works proposed to address security problems based on log files cannot identify code's behaviors. Inspired by the puzzle of whether the source code of an application or job are unique enough, such that it can be identified from the performance logs generated by the system, DeMasi et al. [8] employed and modified the rule ensemble method to predict what source code was running based on the generated performance log files. The Integrated Performance Monitoring (IPM) logs used in the work are collected from a broad set of applications at the NERSC facilities.

Extensive works have shown various structured patterns in performance logs of HPC systems. Orianna et al. [8] proposed a method using the Rule Ensemble method to identify a code by its performance logs based on supervised machine learning. Through interpreting the resulting rule model, it can tell users which components of a code are the most distinctive and useful for identification. The proposed method can monitor the performance of applications

running on HPC resources. Thus, the unapproved code or jobs or unintended usages of HPC systems can be detected.

3.3 ReLog: A log-based Intrusion Detection Framework

We name the proposed anomaly/intrusion detection framework in HPC systems through LOG analytics based on REinforcement learning techniques as ReLog. Motivation and design details are articulated in this section. We first give the overall architecture of ReLog as shown in Figure.3.4. Details about the ReLog will be elaborated in following subsections.

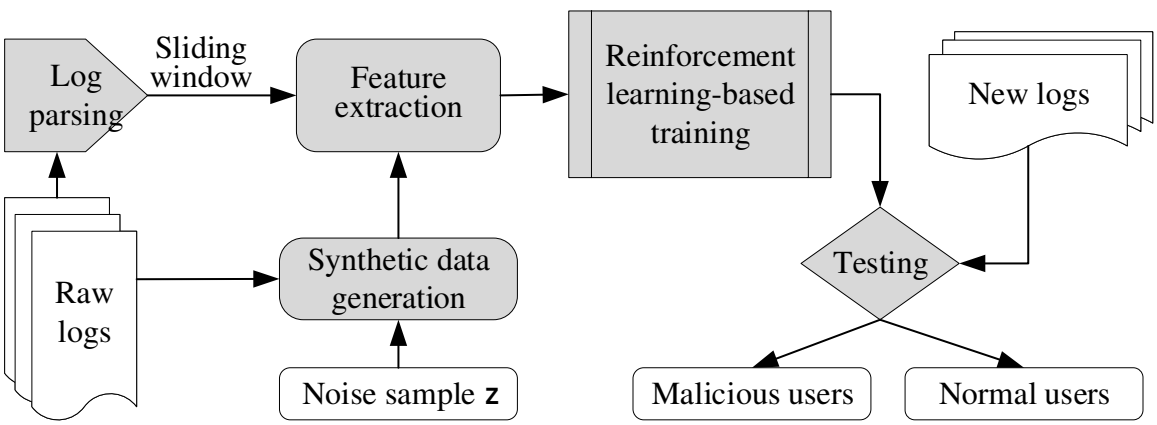


Figure 3.4. Overall architecture of ReLog.

3.3.1 Motivation

The Message Passing Interface (MPI) is a communication protocol that specifies how HPC passes information among computing nodes and clusters. There are various implementations of MPI, including MPICH, Open MPI etc., and SKaMPI is a benchmark that measures the performance of an MPI implementation on a specific hardware. It is an advanced methods for measurements, especially for collective operations of MPI [145].

Unlike traditional classification problem in which they have fixed input format, log analytics in HPC systems is a streaming process as the generation of logs from a running system

is continuous. The streaming logs can be parsed into tokens, which is a small group of lines from logs, further using the tokens to extract feature vectors. As we are detecting anomalous users from their behavior history, i.e., whether an evaluated user is anomalous or not depends not only the pattern of all feature vectors, but also the order. We can treat the detection process as a sequential decision process, the final decision depends on the cumulative decisions from each feature vector and from each transition of two feature vectors.

Reinforcement learning in essence is a sequential decision process. The ultimate goal of reinforcement learning is to achieve the maximum reward at the end of all decisions. In log analytics of HPC systems, what we are interested in is also the final decision after analyzing all of the feature vectors. The similarity of these two problems inspires us to find a solution to detecting anomalous users through log analytics in HPC systems through reinforcement learning techniques.

3.3.2 Design of ReLog

The basic idea of ReLog is to treat the anomaly/intrusion detection based on logs of HPC systems as a reinforcement learning problem. As we observed that in MPI logs, the MPI operations used by computing nodes are usually a subset of pre-defined operations [8], thus we can build feature matrix and treat the feature vectors as states in reinforcement learning. Using the idea of sliding window, we can parse log files of MPI logs into a series of states. Therefore feature vectors extracted from logs can be employed as a state transition process in reinforcement learning. Thus value function algorithms [115] can be employed to obtain the ultimate reward, and further to classify the evaluated users.

ReLog framework has two steps: training and test. We first train the reinforcement learning model based on feature vectors and then test the new logs on the trained model to classify normal and anomalous users.

In the training process, we first build the state space S from logs through counting frequencies of each operation. Each dimension of feature vector denotes one type of MPI operation. The type information of MPI operation is pre-defined [8]. To reduce complexity or dimensions of feature vectors, we can only include information of the top n most occurred operations rather than all. Because jobs running on HPC systems usually are data-intensive jobs, which requires frequent information exchange between computing nodes, and no exception for malicious attackers. Therefore it opens the feasibility of using on the top n most occurred MPI commands to diagnose the running status.

Besides the feature vector dimension, we also need to assign appropriate size of sliding windows, such that the frequency information can be counted within that window. The size of sliding window should reflect dynamics of the job running on HPC systems; thus a too large or too small size of the window will make the training and test steps too sensitive or too insensitive.

When the examined feature vectors are detected as behaviors from an anomalous user, we set this scenario has the maximum reward. Contrarily, when the feature vectors are classified as behaviors from a normal user, we set the reward as 0, i.e., the minimum reward. Thus the training process can be treated as a procedure learning the reward of each state transition. In the testing process, we'll get a cumulative final reward from the state transitions of the testing feature vectors. Based on the reward we can classify the feature vectors of the evaluated logs into normal or abnormal user.

3.3.3 Synthetic Data Generation

In real world log analytics of HPC, usually it is not easy to obtain sufficient training data, especially malicious training data for ReLog. Other cases that require synthetic data generation include (i) when the training data is poorly sampled, i.e., it focuses only on some specific regions, while samples from other regions are necessary to train an efficient and

general model; and (ii) when required training data cannot represent what we are looking for, or worse, it is twisted or noised by some unknown reasons.

Synthetic data generation is a critical step in the training of ReLog, there are many ways have been proposed to address data generation problem [146, 147]. In ReLog, we need to generate two kinds of data, both normal data and abnormal data when the training data is limited. Generating synthetic data based on existing available data is comparatively easier than when there is no available data at all.

In case where no malicious data is available, we adopt a Gaussian-based sampling method to generate malicious data based on available normal data. The two parameters needed are the mean value and the standard deviation. The probability density distribution of the Gaussian distribution is denoted as:

$$p(\mathbf{x}) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{(\mathbf{x}-\mu)^2}{2\sigma^2}}, \tag{3.1}$$

in which $\mathbf{x}$ is the feature vector data. Through changing the mean value and the variance of normal data, we can generate anomalous feature vectors that are required in the ReLog through sampling from the modified Gaussian distribution.

Given the available feature vectors, both normal and abnormal, we plan to construct a Generative Adversarial Network (GAN) [147] based synthetic data generator to generate more normal and abnormal feature vectors to feed the training step in reinforcement learning. In GAN, there are two players: a generator G and a discriminator D. Let p_{data} denotes the distribution that feature vectors are drawn from. The generative model G aims to generate a probability distribution p_g over feature vector data $\mathbf{x}$, which is an estimate of p_{data}. Typically, the generator and discriminator are represented by two deep neural networks. The objective

of the GAN framework can be shown as

$$\begin{aligned}\min_G \max_D V(D,G) =& \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}[\log\ D(\mathbf{x})] \\ &+ \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}(\mathbf{z})}}[\log\ (1 - D(G(\mathbf{z})))],\end{aligned} \tag{3.2}$$

in which $p_{\mathbf{z}}(\mathbf{z})$ is a prior on input noise variables.

The GAN let two players (G and D) play against each other in a game. In ReLog, the feature vector data can also be generated through the generative model. We train two deep neural network models, generative model G and discriminator model D. The two models will play the game to optimize their own objective function. However, to avoid the problem of finding an exact Nash equilibrium, which is challenging in real world, here we use the accuracy of the generated data in discriminator D as a stop requirement, which means after the training process, when the generated data from G can have a higher probability than the pre-set threshold to be misclassified by D, the game stops. The probability is estimated by the percentage of the generated data that are misclassified by D into the real dataset samples in all the generated data.

3.3.4 Experimental Results and Analysis

We collect real-world MPI dataset from HPC systems and validate ReLog framework in this section. Detailed experimental results are given and analysis are specified.

In our work, we use SKaMPI to collect the traces of basic collective operations, such as MPI_Bcast, MPI_Reduce, and so on. According to the communication process of whether to consider the number of processors, we divide the collective operations into two parts, processors-associating (MPI_Scatter, MPI_Gather, etc.) and non-processors-associating (M-PI_Bcast, MPI_Redu, etc.). Thus, we collect different processors and counts about dif-

ferent collective operations, such as MPI_Allreduce, MPI_Allgather, MPI_Alltoall, MPI_Bcast, MPI_Gather, MPI_Reduce, MPI_Scam, MPI_Scatter etc.

In our experimental validation, we mainly focused on four types of logs from directories of Isend_Rev, Send_Irecv, Send_Recv and Ssend_Recv. Each of the directory has 24 log files, 96 log files in total are used in our experiments.

We first analyze statistical properties of the collected dataset, which is the foundation to construct feature vectors. We chose 6 logs from the dataset and show frequency information of MPI operations in Figure. 3.5. The overall frequency information of all logs are shown in Figure. 3.6. From the frequency information depicted in the two figures, we can observe that only a small number of MPI operations (around 30%) are frequently executed, which paves the path for the application of reinforcement learning frameworks.

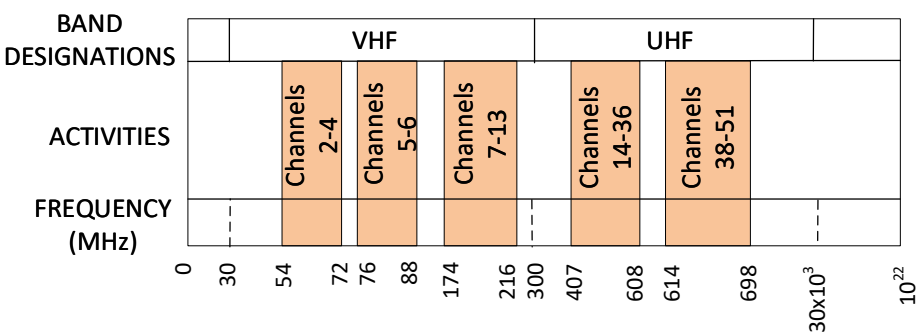


Figure 3.5. The frequency information of the training logs

We divide the collected dataset, which has 4×24 log files, into two categories. Each category has 4×12 log files. One category of logs is used for training, while the other for testing. Initial anomalous data are generated using Gaussian-based sampling method, we generate sufficient anomalous training and testing feature vectors based on the normal training and testing data through changing the mean value of each dimension of the feature vectors while maintain the same variance of training and testing data. We set the sliding

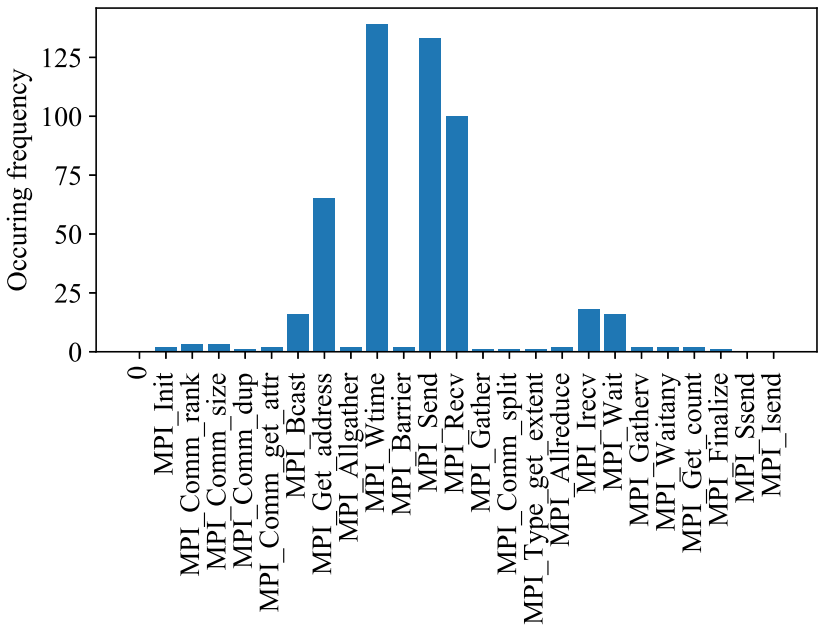


Figure 3.6. The overall frequency information of all the training logs

window size as 150 to form feature vectors in this group of experiments. In Figure. 3.7, we give the relationship between the mean square error (MSE) and the training iterations of GAN-based generation framework. The experimental results show that at least 800 iterations are needed in order to stabilize the MSE.

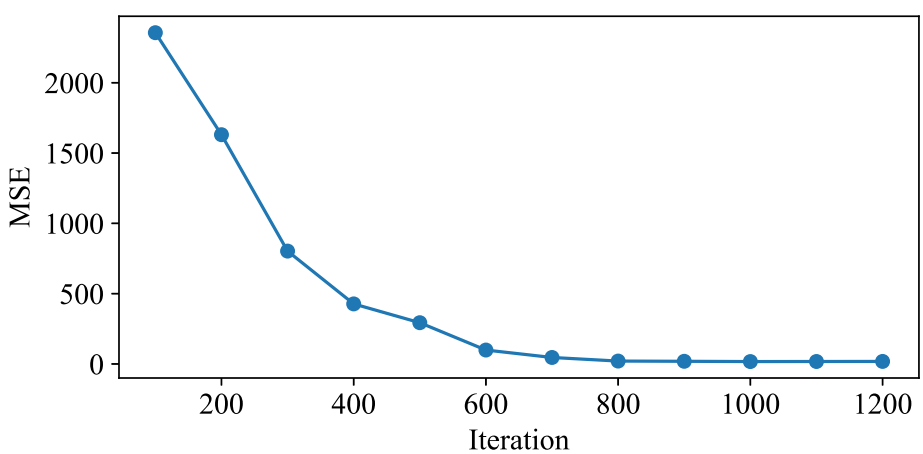


Figure 3.7. Relationship of MSE and training iterations.

Table 3.1. Relationship between sliding window size and detection accuracy.

Window size	**100**	**120**	**140**	**160**	**180**	**200**	**220**
Detection accuracy	0.36	0.42	0.54	0.78	0.93	0.93	0.93

To train an effective ReLog model, we need to choose an appropriate sliding window size to form feature vectors. In this group of experiments, we compare the performance of ReLog under different size of the sliding window. The relationship between the sliding window size and detection accuracy is illustrated in Table 3.1. It demonstrates that when the window size is too small, the performance of ReLog in detecting malicious users will suffer. However, as the sliding window size increases to a point (size of 180 in our experiments), the performance of overall detection accuracy tends to be stable, that's because when the sliding window reaches that point, the information contained in each feature vector already can encode most of the classification information.

To demonstrate the performance of ReLog and show the advantage of using reinforcement learning techniques to detect anomaly/intrusion attackers. We compared ReLog with existing other anomaly/intrusion detection methods in terms of detection accuracy and the time complexity. The methods we compared include: DeepLog [4], which is a deep neural network model employing Long Short-Term Memory (LSTM) to treat logs as natural language sequences; Support Vector Machine (SVN) method, an well-known supervised classification method, which is used in [5] to detect anomaly patterns of logs.

Detailed experimental results on our collected dataset is shown in Table 3.2. We can observe from the results that SVM based detection method has the smallest time cost, that's because the training process involves only finding support vectors, which has been well-solved and the complexity is already optimized. DeepLog and ReLog has similar time cost as a result of training deep learning models in both frameworks. ReLog is especially time-consuming due to the training of multiple deep neural networks in reinforcement learning

Table 3.2. Comparison of ReLog with other existing methods

Detection methods	**Time cost (seconds)**	**Detection accuracy**
DeepLog [4]	56	0.91
SVM [5]	13	0.86
ReLog	107	0.93

frameworks. However, ReLog has the best detection performance than both DeepLog and SVM, which we believe also contributed to the reinforcement learning models, which refines the detection process into a sequential decision process. The experimental results demonstrate the promising future of reinforcement learning techniques in detecting anomalous users in HPC systems.

3.3.5 Future Work

The security of HPC systems will experience more challenges in the future especially as the widely applications of machine learning techniques, both in terms of attacks and defenses. Here we give several potential research topics as future work:

- There are various methodologies proposed to defend intrusion attacks in HPC systems based on log files. Through reviewing many of these methods, we found that most of the methods are focused on a small part of the log files, i.e., there still lacks a strategy that can utilize all the available logs that employing state-of-the-art machine learning techniques to study the security problem. We believe this would be a promising research frontier.

- Deep learning techniques are widely used in log file analytics to classify different types of users by their application behaviors. It is worth noting that different platforms will have different user behaviors and data types, thus lead to different feature extraction

methods. The efficient and effective feature vector extraction methods are at the core of applying deep learning techniques in future work.

- If we look beyond security problems in HPC systems, we observed that the problem faced by other distributed systems, such as cloud computing, cluster computing and grid computing, are similar with the problems we faced in HPC systems. This is because they all have similar architectures and utilities. Security solutions in cloud computing, cluster computing, grid computing and many other distributed computing models [120, 128] can be transferred to the HPC systems due to their inner similarities and this will be a promising future research topic.

The security of HPC systems has drawn emerging attention from both academia and industry. With the rapid growing of HPC systems in terms of both scale and complexity, machine learning based security inspection, evaluation and malicious behavior detection will play a more practical role in improving the usability and security of HPC systems.

3.4 Binary Code Similarity Detection

We elaborate the binary code similarity detection based on machine learning techniques, more specifically LSTM and Siamese Neural Networks in this section.

3.4.1 Background

We give the background information of the LSTM networks, Siamese neural networks and the related works of binary code similarity detection.

3.4.1.1 LSTM Recurrent Neural Networks

One of the main advantages of recurrent neural networks is they have loops within their structure that allowing information to persist. However, ordinary RNN models have van-

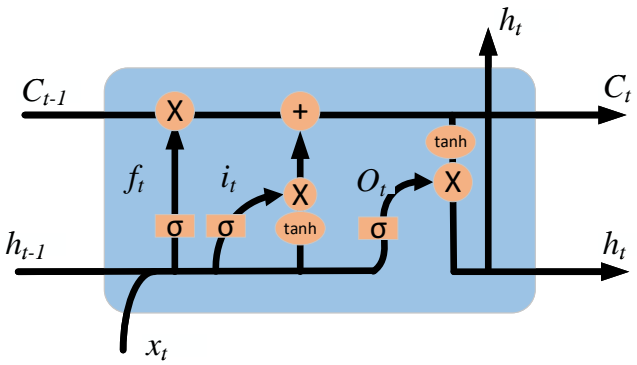


Figure 3.8. The repeated module in a standard LSTM networks

ishing gradients and exploding gradients problems[148], LSTM offers a decent solution to these two problems through using constant error carousels, which could protect and control the cell state. As a special kind of RNN, LSTM has the capability of learning long-term dependencies. The basic form of LSTM consists of a chain of repeated neural networks. The structure of the repeated module is shown as in Figure. 3.8. In the module, x_t is the input at time t and h_t denotes the corresponding output value. C_{t-1} represents the cell state, which runs straight down the entire LSTM model.

At the first part of the LSTM modules, it aims to decide what kind of information in the cell state from previous time will be deleted through a Sigmoid layer, mathematically it is shown as:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f), \tag{3.3}$$

in which W_f, b_f is the parameters of the corresponding neural network layer (consists of a bunch of Sigmoid functions) and h_{t-1} is the previous output.

The following part of LSTM module is to decide what kind of new information we need to add to the cell state. The operations within the repeated model could be shown mathe-

matically as:

$$\begin{aligned} i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i), \\ g_t &= \tanh(W_g \cdot [h_{t-1}, x_t] + b_g). \end{aligned} \tag{3.4}$$

Similarly, W_i, W_g, b_i, b_g are the corresponding neural network layer parameters.

After the two operations on the cell state, the new cell state is updated through a pointwise operation from f_t, i_t and g_t through:

$$C_t = f_t * C_{t-1} + i_t * g_t, \tag{3.5}$$

which includes both the added information from current time and also compromised some historical information. The output of the LSTM is denoted as:

$$h_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) * \tanh(C_t), \tag{3.6}$$

in which $\tanh(C_t)$ is used to push the cell cell state values to be between -1 and 1.

LSTM networks have been widely applied to various real-world applications and achieved state-of-the-art performances such as speech recognition [149], handwriting recognition [150]. In [151], LSTM networks are used to process localized features and perform the classification to detect stealthy malware. In this work, we employed LSTM to extract general information from local features of each block in disassembled CFGs of binaries and further use the information for similarity detection.

3.4.1.2 Siamese Neural Networks

Siamese neural networks were first introduced by Bromley and LeCun in the early 1990s, in which Siamese neural networks are designed for verification of signatures written on a pen-

input tablet [152]. It has been successfully applied in image recognition [47], gait recognition [153].

Siamese neural networks consist of a class of neural network architectures, in which they have two or more identical sub-networks, each of them has the same configuration in terms of the parameters and weights. In the training process, the parameter is updated concurrently across all sub-networks. Their values will be mirrored thus all the parameter values across different sub-networks will still be the same. This kind of architecture can be used to compare the similarity of different inputs.

Siamese neural networks have some impressing advantages compared to tradition neural networks [154, 155]. For example, Siamese neural networks are more robust to class imbalance as new classes of data can be added to the network without training the whole model again. Siamese networks focus on learning embedding (in the deeper layer), further the same classes/categories of the inputs are placed close together, which we denote it as semantic similarity. Another important advantage is that Siamese networks can be easily trained using standard machine learning techniques on pairs sampled from source data and provide a competitive approach that does not rely upon domain-specific knowledge.

The training process of Siamese Neural Networks involves training a pairwise model, thus traditional entropy loss cannot be used in this case. Triplet loss is a usual loss function used in Siamese neural networks, in which a baseline (anchor) input is compared to a positive (truthy) input and a negative (false) input. The optimization objective of the training is to minimize the distance between the baseline input and the positive input while maximizing the distance between the baseline input and the negative input, as shown in Figure.3.9.

Mathematically the triplet loss can be formulated as:

$$L(a,p,n) \;=\; \frac{1}{2}\{max(0, m + D^2(a,p) - D^2(a,n))\}, \tag{3.7}$$

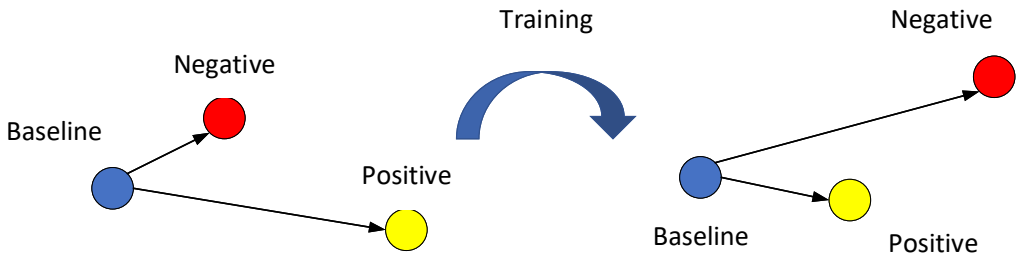


Figure 3.9. The triplet loss of Siamese Neural Networks.

in which $D(u, v) = |u - v|_2$, and m is the desirable distance for dissimilar pair (p,n).

Another typical type of loss function is the contrastive loss. The logic behind contrastive loss is similar to the triplet loss. Contrastive loss is used to learn embedding in which two similar points have a smaller Euclidean distance while two dissimilar points have a large Euclidean distance. Given an input training pair (x_1, x_2), we have the label: $y = 0$ if (x_1, x_2) is similar and 1 if (x_1, x_2) is dissimilar pair. The optimization objective in terms of contrastive loss can be written as:

$$min\ L(x_1, x_2) = \frac{1}{2}(1 - y)D^2 + \frac{1}{2}y\ max\{0, m - D\}^2, \tag{3.8}$$

in which D is the Euclidean distance of the outputs from the two networks given inputs x_1, x_2.

Given the advantages of Siamese Neural Networks, the embeddings we obtained through LSTM neural networks could be fed into the Siamese Neural Network to perform similarity measurement such that the embedding information could be fully considered.

3.4.1.3 Binary Code Similarity Detection

Binary code similarity detection is quite challenging while also rewarding. It is challenging because the binaries generated from software projects could vary enormously due to the diversity of processors, compilers, optimization level options, and platforms. It is rewarding

because we could evaluate the security of the software without the source code, which often is unavailable.

Many of existing works regarding to binary code similarity detection follows a similar logic [2, 1]. It first recovers CFGs from the binaries and then based on the statistical and structural information of the CFGs to perform graph matching. To achieve scalability and high accuracy concurrently, graph-embedding based methods such as [43] focused on learning an indexable feature representation from the CFGs. Based on the embeddings of test binaries and the embeddings of known bugs or benchmarks, a similarity score could be measured, further paves the way for evaluating the security vulnerabilities.

A basic block distance-based method, named as discovRE, is proposed in [2]. The general idea behind discovRE is to calculate similarity between functions based on CFGs disassembled from binaries. However, this process is computationally expensive if not being processed appropriately. To minimize the computation cost, discovRE employs an efficient pre-filter to identify a small set of candidate functions and then use them to search for similar functions in the binaries that need to be evaluated.

Another type of methods is the Neural network-based methods [43, 1, 44], which focuses on using neural networks to embed CFGs. The main advantage of this type of methods is they could alleviate the limitations of graph matching-based approaches, such as high computation cost. Further the embeddings can be used for malware detection or classification.

Grieco et al [41]. proposed a vulnerability discovery tool, named VDiscover. They combined static and dynamic analysis with machine learning techniques to predict the vulnerabilities among binaries. The basic idea of the approach is that it applies both static and dynamic analysis to extract features from a large-scale of binary programs. These features are further studied and used to predict vulnerabilities by machine learning techniques.

In this work, we propose a new embedding method based on LSTM networks, which has the advantage of containing the general information of all the previous inputs and the

dependency information. Thus, the feature vectors of each block within the CFGs could be processed and embedded into an indexable representation. Our experimental results show that this method is computationally efficient compared with the existing graph neural network-based methods[1].

In addition, our method of using Siamese Neural Network to measure the similarity based on the cell state of the LSTM network is different from existing studies [152, 47, 153]. To the best of our knowledge, we are the first to use the cell state of LSTM in CFG embedding and use Siamese Neural Network to measure the similarity of such embedding, which has the advantage of measuring the semantic similarity of the overall local feature information of CFGs.

3.4.2 Our Proposed Binary Code Similarity Detection Framework

We elaborate the motivation and the proposed binary code similarity detection framework using LSTM and Siamese neural network in this section.

3.4.2.1 Motivation

Existing methods of performing binary code similarity detection mainly focused on two perspectives: graph matching-based methods and deep neural network-based methods. Although graph matching-based methods is widely discussed in binary code similarity detection [2, 43]. The shortcomings are also obvious [1]: (i) It is less flexible for the similarity functions approximated through graph matching techniques to adapt to other applications; (ii) The graph matching algorithms suffer from low efficiency, thus leads to the inefficiency of the similarity detection process.

Another perspective is the deep neural network-based methods [36, 41, 44, 1], which often involves training a deep graph neural network or semantic-aware neural network. This could

complicate the problem especially in terms of computation cost when the input dimensions increase.

In the similarity measurement stage, the traditional methods including Euclidean distance comparison and classification methods focus on the statistical characteristics of two inputs. However, the embedding we extracted from graph-based embedding methods and neural network-based methods contain much information that traditional methods fall short of capturing. The embedding information of the binaries requires a more effective method to perform the similarity comparison.

Based on the above points identified, we propose an LSTM network-based embedding methods in this work to improve the information representation in the embedding process, and we employ Siamese Neural Networks to compare the similarity between different binaries. This strategy does not require graphs, which avoids the computation cost concerns from graph computation.

The detailed architecture design using LSTM networks to perform embedding and Siamese neural networks to conduct similarity comparison is described in the following subsection.

3.4.2.2 Our Proposed Framework

Our proposed method includes three steps: local block feature extraction, LSTM network-based embedding and Siamese neural network-based similarity detection.

To perform binary code similarity detection, we need to first extract corresponding characters from binary files and transform these characters into a matrix or vector representation. In this work, we use angr [156], a multi-architecture binary analysis toolkit, to disassemble the binaries and get the desired CFGs. Further we could perform embedding based on the CFGs.

angr is capable of analyzing binaries in both static and dynamic disassembling styles. Therefore there are two types of CFGs that can be disassembled: static CFGs (CFGFast)

and dynamic CFGs (CFGEmulated). CFGFast uses static analysis to generate the CFGs of binaries. It is faster but might lose a small number of control-flow transitions which can only be resolved at execution time. CFGEmulated generates CFGs through symbolic execution, which is usually slower, especially for large binaries. In our experiment, we found that usually CFGFast achieves better performance in terms of computation cost.

The disassembled files recovered through the CFGFast operation of an example binary is shown as in the following, we disassembled the binary file in a depth-first order, as shown in Listing 3.3:

Listing 3.3. The disassembled files recovered from binary files.

```
<CFGNode _init [11]>
0x80489c4: push ebp
0x80489c5: mov ebp, esp
0x80489c7: sub esp, 8
0x80489ca: call 0x8048c90
offsprings: 1
betweenness: 2.0495163141498604e-05
<CFGNode call_gmon_start [27]>
0x8048c90: push ebp
0x8048c91: mov ebp, esp
0x8048c93: push ebx
0x8048c94: push eax
0x8048c95: call 0x8048c9a
0x8048c9a: pop ebx
0x8048c9b: add ebx, 0x404a
0x8048ca1: mov eax, dword ptr [ebx + 0xac]
0x8048ca7: test eax, eax
0x8048ca9: je 0x8048cad
offsprings: 2
betweenness: 3.415860523583101e-05
```

From the above disassembled information, we can observe the detailed graph node, or block information including the operations, offspring and betweenness information, which can be used to describe the characteristics of the binaries. As the results show, we obtain the disassembled information of each CFG basic block (in a depth-first order), including the opcode and the statistic results of offspring number and betweenness count, which combined can be used to comprehensively describe the characteristics of the binaries.

Given a binary function, the disassembled blocks contain multiple attributes that could be utilized to characterize the corresponding binary function. The basic block attributes can be divided into two types according to their level[2, 1]: block-level attributes and inter-block level attributes. Block-level attributes includes No. of String Constants, No. of Numeric Constants, No. of Calls, No. of Transfer Instructions, No. of Data Transfer Instructions, No. of Logic Instructions, No. of Arithmetic Instructions, etc. Inter-block level attributes include No. of offsprings, betweenness, etc.

Here we employ an intuitive statistical property, i.e., the total number of times a type of operation occurred, together with two inter-block attributes, to describe the block.

Given above statistical and structural attributes of the block, we form them into a 9-dimension feature vector for each block (a.k.a., graph node). In the following part we'll explain how to transform the local features of all blocks from a binary file into an indexable embedding.

There are dozens, if not hundreds or thousands, of blocks could be disassembled given a binary file. Correspondingly we could obtain the same number of localized feature vectors. However, how to learn an indexable embedding that has the capability of reflecting the overall characteristics of the binary based on all these local features?

Existing methods include graph-embedding methods [2, 43] focused on transform the CFGs into a graph, and then using graph matching-based method such as bipartite matching algorithm to calculate the similarity of two CFGs. Deep graph neural network-based methods [43, 1, 44] perform the embedding through deep neural networks. However, LSTM networks offers a practical mechanism to potentially store the related information of all historical inputs. Given a sequence of inputs, the input x_t at each time slot t will update the cell state C_t based on the learned parameters of the neural network layer.

The update process at each time slot t will includes two steps: the first step is to decrease or delete some unrelated information from the previous cell state through a "forget gate

layer". Mathematically it is denoted as:

$$C_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) * C_{t-1} \tag{3.9}$$

In our local feature embedding scenario, this operation could be utilized to compromise the influence of some extreme blocks, and also, we have the capability of getting rid of redundant information through the training of the neural network layers.

The second step of the update process is to add information from each input to the cell state C_t, also known as the "input gate layer", which is shown as:

$$C_t = C_t + \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) * \tanh(W_g \cdot [h_{t-1}, x_t] + b_g). \tag{3.10}$$

After the two steps stated above, the information of interest with regard to the binary embedded in the features extracted at each local block will be able to get filtered and reflected by the cell state if trained appropriately. Therefore, after training the parameters within the LSTM network with the training dataset, a final cell state will be learned on each type of binaries. This cell state is learned over the entire training blocks. We use this cell state as the final indexable feature embedding of the corresponding binary file.

Given the obtained indexable feature embeddings of binaries, we could apply various traditional methods such as Euclidean distance, neural networks to do similarity measure or classification. However, most of these traditional methods focus on the numeric features the embeddings presented while falling short of denoting the semantic information.

Provided with the multiple advantages of Siamese neural network [47] including: **a)** More robust to class imbalance; **b)** learning from semantic similarity of the embeddings; **c)** easily to be trained using standard optimization techniques. We design a Siamese neural network-

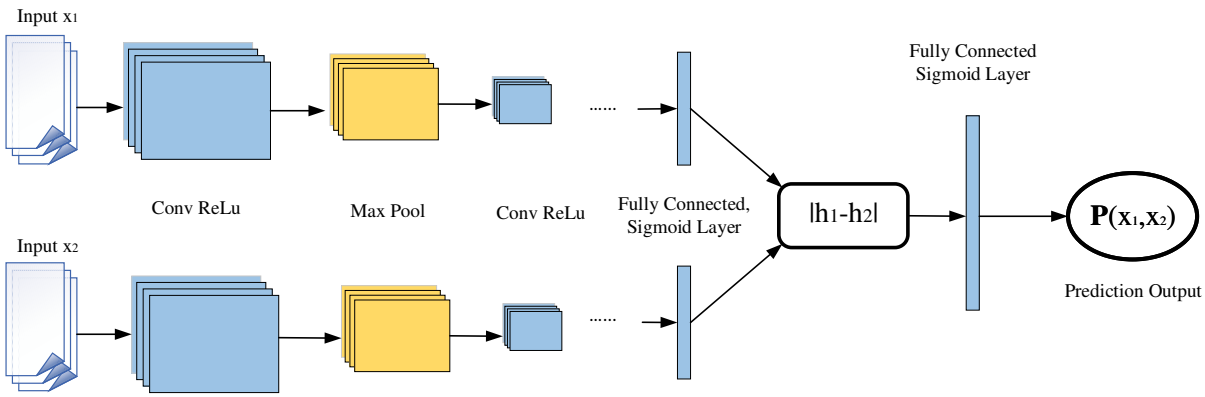


Figure 3.10. The architecture of binary code similarity detection.

based similarity detection mechanism to compare the embeddings from the test binaries with embeddings of existing known malwares or other trained binaries.

Our designed Siamese neural network-based binary analysis framework is shown in Figure.3.10. The architecture includes a sequence of convolutional layers, max pool layers, fully connected layers. Each of the neural network layer uses a single channel with filters of varying size and the stride is fixed to 1. We employ rectified linear Unit (ReLu) activation functions in each layer. The max pooling operation is assigned with a filter size of stride 2.

We have two inputs for the model, each of which is an embedding we learned through LSTM networks. We aim to minimize the distance of the outputs of each sub-model in the training process if the two inputs are from the same category, while maximizing the distance of the outputs of each sub-model if the two inputs are from different categories.

The units in the last convolutional layer are flattened into a single vector. A fully connected layer is followed after this convolutional layer. Then the induced distance is computed based on the output (h_1, h_2) of the previous fully connected layer, further it is given to a single sigmoid activation function layer for prediction. Mathematically, the

prediction vector is shown as:

$$\mathbf{P}(x_1, x_2) \;=\; \sigma(\sum_j \alpha_j |h_1^{(j)} - h_2^{(j)}|), \tag{3.11}$$

in which x_1, x_2 are the two input embeddings. σ denotes the sigmoidal activation function. α_j are parameters learned by the model during the training process, which is used to weight the importance of the component-wise distance. The final layer induces a measure on the learned feature space of the previous hidden layer and scores the similarity between the two feature vectors.

Start from this architecture, we can first train the model on our known training dataset, then when new test binaries come in, we only need to compare the test input with the reference input (which comes from the trained dataset). Another big advantage of this architecture is that when we have new samples, we only require a very small number of samples to be stored in the dataset, using it as a reference, we can calculate the similarity for any new test instance with this type of samples [47].

In the learning process, we use M to denote the minibatch size and i is used to index the i_{th} minibatch. Let $\mathbf{y}(x_1^{(i)}, x_2^{(i)})$ be a vector of length M that contains the labels for the minibatch. We set $y(x_1^{(i)}, x_2^{(i)}) = 1$ whenever x_1 and x_2 are from the same category and $y(x_1^{(i)}, x_2^{(i)}) = 0$ if they are from different categories. Then the regularized cross-entropy objective on the binary classifier scenario takes the following form:

$$\begin{aligned} \mathcal{L}(x_1^{(i)}, x_2^{(i)}) = \; & \mathbf{y}(x_1^{(i)}, x_2^{(i)}) \; log \; \mathbf{P}(x_1^{(i)}, x_2^{(i)}) + \\ & (1 \; - \; \mathbf{y}(x_1^{(i)}, x_2^{(i)})) \; log \; (1 \; - \; \mathbf{P}(x_1^{(i)}, x_2^{(i)}) \;) + \\ & \lambda^T |\mathbf{w}|^2, \end{aligned} \tag{3.12}$$

in which $\lambda^T |\mathbf{w}|^2$ is the regularization term.

In terms of the update policy, we employ the standard backpropagation algorithm to optimize the parameter values. Let η_j denotes the learning rate and μ_j the momentum. The update rule at epoch T can be formularized as:

$$\begin{aligned} \mathbf{W}_{kj}^{(T)}(x_1^{(i)}, x_1^{(i)}) &= \mathbf{W}_{kj}^{(T)} + \Delta\mathbf{W}_{kj}^{(T)}(x_1^{(i)}, x_1^{(i)}) + 2\lambda_j|W_{kj}|, \\ \Delta\mathbf{W}_{kj}^{(T)}(x_1^{(i)}, x_1^{(i)}) &= -\eta_j \nabla w_{kj}^{(T)} + \mu_j \Delta\mathbf{w}_{kj}^{(T-1)}. \end{aligned} \tag{3.13}$$

In which $\nabla w_{kj}^{(T)}$ denotes the partial derivative in terms of the weight between the j_{th} neuron in one layer and k_{th} neuron in the following layer.

In our following experiments, all the values of parameters within the convolutional neural network is assigned with a normal distribution of zero-mean and a standard deviation of 0.01. Bias is initialized with also a normal distribution of mean 0.5 and standard deviation 0.01. The initialization configuration is similar with those in [47] due to their similarity in terms of objective.

3.4.3 Experimental Results and Analysis

To validate the performance of our proposed framework, which using LSTM networks to embed the general features of binary files based on the local features of disassembled CFGs, and employing the Siamese neural network to perform the similarity detection, we experimented the framework on a dataset we collected from real-world scenarios. The dataset includes 4000+ ELF malware executables. It contains 560 categories of the binaries, and each category contains multiple binary files.

In the experiments, we implemented the framework based on TensorFlow v2.4.1 and Keras 2.4.3. The experiments are run on a Ubuntu 18.04 version. The framework is implemented using Python 3.6.9.

3.4.3.1 Local Feature Vector Extraction

We first perform the disassembling operations on the binaries to obtain the corresponding CFGs. Then based on the graphs, we could extract the local block feature vectors, in our experiments, we build the local feature vectors as shown in Equation 3.14:

$$
\begin{aligned}
[\ & \text{No. of string constants,} \\
& \text{No. of numeric constants,} \\
& \text{No. of arithmetic instructions,} \\
& \text{No. of logic instructions,} \\
& \text{No. of transfer instructions,} \\
& \text{No. of calls,} \\
& \text{No. of data transfer instructions,} \\
& \text{No. of offspring,} \\
& \text{the value of betweenness}\ \ ]^{\intercal}.
\end{aligned}
\tag{3.14}
$$

This is intuitive and easy to obtain from the disassembled CFGs, and also is widely used in existing methods[2, 43].

Based on the assigned rule, we could easily obtain the numerical forms of the local feature vectors for each block in a binary file. An examplified numerical features of a sequence of local blocks is shown as in Listing 3.4:

Listing 3.4. An examplified numerical features of a sequence of local blocks.

```
                ......
4 0 0 0 1 1 1 2.0495163141498604e-05
1 10 1 0 0 1 2 2 3.415860523583101e-05
1 3 0 0 0 0 1 1 6.968355468109526e-05
0 1 0 0 0 1 0 1 0.0001257036672678581 1
1 6 0 0 0 0 2 2 6.0119145215062575e-05
0 2 0 0 0 0 0 1 4.9188391539596654e-05
0 1 0 0 0 1 0 1 0.00018172377985462097
1 8 0 0 0 0 3 2 6.968355468109526e-05
0 4 0 0 0 0 0 1 2.4594195769798327e-05
0 2 0 0 0 0 0 1 0.00011750560201125867
```

. . . . . .

The local feature vectors extracted above represent both the block-level attributes information and inter-block level attributes information. As we can observe from local feature vectors, they are changing at each block. To obtain an indexable representation of the binary that contains multiple blocks, we feed this sequence of local feature vectors into the LSTM networks.

3.4.3.2 LSTM Network-based Embedding

To embed the obtained local feature vectors, we first need to truncate the blocks we have as we want to train each of the binary files on the same number of local features. However, the real-world binaries we collected are vary each other in terms of the length. We adopt a strategy that choose a fixed number of the blocks from each type of the binary file in a random way but without disrupting the order of the chosen sequence.

We form the embedding problem as a regression problem, in which the input is the feature vector at time t, while the output is the feature vector at time $t + 1$. However, to make the prediction process could take more of the past time slots into consideration, we employ multiple previous time slots as the input to predict the following feature vector of the block. We found that if too many previous time slots are considered, then the prediction output will be less reflective regarding the variation and the number of training samples will decrease dramatically. There is a tradeoff balance between the flexibility and accuracy.

In the experiment, we employ 5 previous feature vectors as the input to train the model, and we set the dropout value as 0.1, and the training epoch is set as 100. At each prediction. The LSTM blocks or neurons we adopt is set as 16. In Figure. 3.11, we show the relationship of the averaged Mean Squared Error (MSE) with the training epochs of three representative binaries, from which we can observe that when the training epochs approaching 70, the

averaged MSEs for all three binaries are become stable. Thus, we could output the hidden cell state vectors as the embeddings for those binary files.

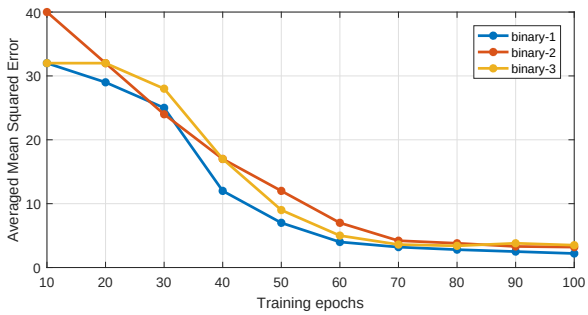


Figure 3.11. The relationship of the training epochs with the averaged MSE.

3.4.3.3 Siamese Neural Network-based Similarity Detection

Table 3.3. The comparison of our proposed framework with existing works [1, 2].

Methods	**Accuracy**	**Training time cost**	**Testing time cost**
Our proposed method	0.85	1.00	1.00
Xu et al.[1]	0.85	1.24	1.15
Eschweiler et al.[2]	0.83	0.78	0.67

After we obtained the embeddings of the binaries, we have the indexable representation of each binary file. Thus, the remaining work is to feed them into a Siamese neural network and perform the similarity detection.

In this group of experiments, we focused on classifying a test binary to the pre-trained binary categories. We chose 10 categories of the binaries from our collected dataset as the training categories. As some categories include only a very small number of binaries, thus we exclude them in our experiment such that we have enough binaries under each category that could be divided into a training set and a testing set. For the testing binaries, we'll include both the binaries from the training categories and from those categories that are not included in the training process except specially specified.

Since in a Siamese neural network, the weights from both sub-networks are supposed to be identical with each other, thus we use only one model and feed two inputs in succession. The optimization process or the backpropagation is conducted after we calculate the loss for two inputs. We build the Siamese neural network includes 3 fully connected convolutional layers with a structure of $16 \times 32 \times 16$, and the dropout value of 0.1.

We first show the experimental results of the training performance using only test data from the categories that we have trained on. We define a measure named as detection accuracy to describe the performance, which is defined mathematically as:

$$\text{Detection Accuracy} = \frac{\text{No. of test binaries rightly classified}}{\text{No. of total test binaries}} \tag{3.15}$$

We use half of the binaries within each category as the training data and the other half as the testing data, the relationship of the detection accuracy with the training epochs is shown in Figure.3.12. From the results we know that after training the Siamese neural network with at least 60 epochs, we can achieve the detection accuracy of around 90%, which validates the performance of the framework.

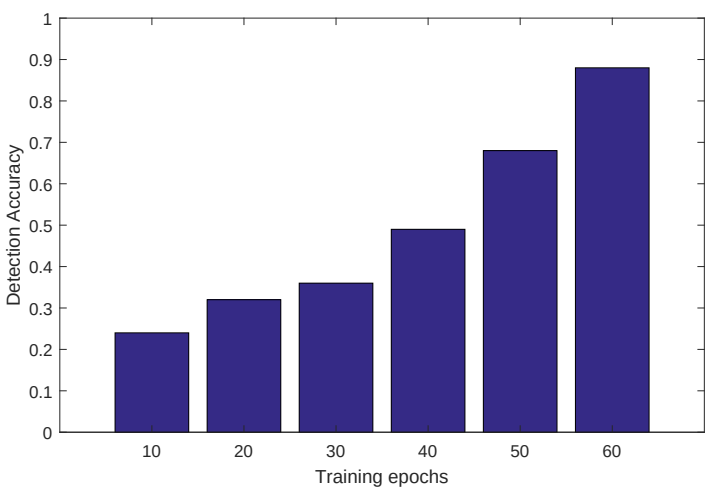


Figure 3.12. The relationship of detection accuracy with training epochs

We also compared our proposed framework with existing methods as shown in [1, 2], which are representative methods to detect malware. [1] focuses on a graph embedding network to convert the graph into embeddings for binary functions, while [2] focuses on the maximum common subgraph isomorphism to measure the structural similarity between two different binaries.

We compared the performance and relative time cost of the three methods and the results are shown in the Table 3.3. Both the time cost for training and testing are compared with our proposed method in a relative way, which means our proposed method is set as a benchmark with the value of 1. From the comparison results we know that our method shows a higher training efficiency while still maintains similar detection accuracy performance with method [1]. Compared with [2], our method showed a slightly better performance though suffering from the training of two neural networks which cost a little bit more time. Compared with our proposed LSTM + Siamese neural network combination, the methods in [1] require training a deep graph embedding neural network, while the methods in [2] require a graph similarity algorithm based on the maximum common subgraph isomorphism.

Chapter 4: Conclusion

In the dissertation, we first explored the partial model problem in cooperative spectrum sensing systems. We proposed a powerful attack mechanism named the LEB attack against cooperative spectrum sensing. Given the gap between existing defenses and new adversarial learning-based attacks, such as LEB attacks, we designed an influence-limiting defense, sided with existing defenses to counter LEB attacks and other similar attacks. Further we presented a low-cost version of the influence-limiting defense by employing the "divide and conquer" strategy to decrease the computation cost dramatically.

Besides, we offered a new perspective through employing reinforcement learning techniques to detect anomalies based on MPI logs in HPC. Extended from this project, we found that binary code similarity detection plays an important role in evaluating the security of a software project that are closed source. Therefore, we proposed an LSTM neural network-based method to obtain an indexable embedding from the dissembled control flow graphs of binary files, and we employed Siamese Neural Networks to conduct the similarity comparison of two embeddings due to that Siamese Neural Networks have the capability of learning the semantic information embedded in the inputs.

In the future work, we will focus on improving the security defense capability of spectrum sensing and also focus on extracting more representative local features from the disassembled blocks of the binaries.

References

[1] Xiaojun Xu, Chang Liu, Qian Feng, Heng Yin, Le Song, and Dawn Song. Neural network-based graph embedding for cross-platform binary code similarity detection. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 363–376, 2017.

[2] Sebastian Eschweiler, Khaled Yakdan, and Elmar Gerhards-Padilla. discovre: Efficient cross-architecture identification of bugs in binary code. In *NDSS*, volume 52, pages 58–79, 2016.

[3] Ramesh Bulusu, Pallav Jain, Pravin Pawar, Mohammed Afzal, and Sanjay Wandhekar. Addressing security aspects for hpc infrastructure. In *2018 International Conference on Information and Computer Technologies (ICICT)*, pages 27–30. IEEE, 2018.

[4] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1285–1298. ACM, 2017.

[5] Shilin He, Jieming Zhu, Pinjia He, and Michael R Lyu. Experience report: system log analysis for anomaly detection. In *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*, pages 207–218. IEEE, 2016.

[6] Pinjia He, Jieming Zhu, Shilin He, Jian Li, and Michael R Lyu. Towards automated log parsing for large-scale log data analysis. *IEEE Transactions on Dependable and Secure Computing*, 15(6):931–944, 2017.

[7] Pingchuan Ma. Log analysis-based intrusion detection via unsupervised learning. *Master of Science, School of Informatics, University of Edinburgh*, 2003.

[8] Orianna DeMasi, Taghrid Samak, and David H Bailey. Identifying hpc codes via performance logs and machine learning. In *Proceedings of the first workshop on Changing landscapes in HPC security*, pages 23–30. ACM, 2013.

[9] Linyuan Zhang, Guoru Ding, Qihui Wu, Yulong Zou, Zhu Han, and Jinlong Wang. Byzantine attack and defense in cognitive radio networks: A survey. *Commun. Surveys Tuts.*, 17, 2015.

[10] Husheng Li and Zhu Han. Catch me if you can: An abnormality detection approach for collaborative spectrum sensing in cognitive radio networks. *IEEE Trans. Wireless Commun.*, 9, 2010.

[11] Ruiliang Chen, J-M Park, and Kaigui Bian. Robust distributed spectrum sensing in cognitive radio networks. In *IEEE INFOCOM*, 2008.

[12] Praveen Kaligineedi, Majid Khabbazian, and Vijay K Bhargava. Malicious user detection in a cognitive radio cooperative sensing system. *IEEE Trans. Wireless Commun.*, 9, 2010.

[13] Ankit Singh Rawat, Priyank Anand, Hao Chen, and Pramod K Varshney. Collaborative spectrum sensing in the presence of Byzantine attacks in cognitive radio networks. *IEEE Trans. Signal Process.*, 59, 2011.

[14] Omid Fatemieh, Ali Farhadi, Ranveer Chandra, and Carl A Gunter. Using classification to protect the integrity of spectrum measurements in white space networks. In *NDSS*, 2011.

[15] Wei Wang, Lin Chen, Kang G Shin, and Lingjie Duan. Secure cooperative spectrum sensing and access against intelligent malicious behaviors. In *IEEE INFOCOM*, 2014.

[16] Qiben Yan, Ming Li, Tingting Jiang, Wenjing Lou, and Y Thomas Hou. Vulnerability and protection for distributed consensus-based spectrum sensing in cognitive radio networks. In *IEEE INFOCOM*, 2012.

[17] Huifang Chen, Ming Zhou, Lei Xie, and Jie Li. Cooperative spectrum sensing with M-ary quantized data in cognitive radio networks under SSDF attacks. *IEEE Trans. Wireless Commun.*, 16, 2017.

[18] Changlong Chen, Min Song, Chunsheng Xin, and Mansoor Alam. A robust malicious user detection scheme in cooperative spectrum sensing. In *IEEE GLOBECOM*, 2012.

[19] Federico Penna, Yifan Sun, Lara Dolecek, and Danijela Cabric. Detecting and counteracting statistical attacks in cooperative spectrum sensing. *IEEE Trans. Signal Process.*, 60, 2012.

[20] K. M. Thilina, K. W. Choi, N. Saquib, and E. Hossain. Machine learning techniques for cooperative spectrum sensing in cognitive radio networks. *IEEE J. Sel. Areas Commun.*, 31, 2013.

[21] Huaxia Wang and Yu-Dong Yao. Primary user boundary detection in cognitive radio networks: Estimated secondary user locations and impact of malicious secondary users. *IEEE Trans. Veh. Technol.*, 67, 2018.

[22] Sutharshan Rajasegarar, Christopher Leckie, and Marimuthu Palaniswami. Pattern based anomalous user detection in cognitive radio networks. In *IEEE ICASSP*, 2015.

[23] Nicolas Papernot, Patrick McDaniel, Arunesh Sinha, and Michael Wellman. Towards the science of security and privacy in machine learning. *arXiv preprint arXiv:1611.03814*, 2016.

[24] Nicolas Papernot, Patrick McDaniel, and Ian Goodfellow. Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. *arXiv preprint arXiv:1605.07277*, 2016.

[25] David Pellerin, Dougal Ballantyne, and Adam Boeglin. An introduction to high performance computing on aws. *Amazon Whitepaper*, 2015.

[26] Sean Peisert. Security in high-performance computing environments. *Communications of the ACM*, 60(9):72–80, 2017.

[27] Ian Foster, Nicholas T Karonis, Carl Kesselman, and Steven Tuecke. Managing security in high-performance distributed computations. *Cluster Computing*, 1:95–107, 1998.

[28] Mark Barnell, Courtney Raymond, Chris Capraro, Darrek Isereau, Chris Cicotta, and Nathan Stokes. High-performance computing (hpc) and machine learning demonstrated in flight using agile condor®. In *2018 IEEE High Performance extreme Computing Conference (HPEC)*, pages 1–4. IEEE, 2018.

[29] Ivan Beschastnikh, Yuriy Brun, Michael D Ernst, and Arvind Krishnamurthy. Inferring models of concurrent systems from logs of their behavior with csight. In *Proceedings of the 36th International Conference on Software Engineering*, pages 468–479. ACM, 2014.

[30] Mathieu Blanc, Jérémy Briffaut, D Gros, and C Toinard. Piga-hips: Protection of a shared hpc cluster. *International Journal on Advances in Security Volume 4, Number 1 & 2, 2011*, 2011.

[31] Scott Campbell and Jim Mellander. Experiences with intrusion detection in high performance computing. *Proceedings of the Cray User Group (CUG)*, 2011.

[32] Ryan L Braby, Jim E Garlick, and Robin J Goldstone. Achieving order through chaos: the llnl hpc linux cluster experience. Technical report, Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States), 2003.

[33] Fabrice Gadaud, Mathieu Blanc, and Frédéric Combeau. An adaptive instrumented node for efficient anomalies and misuse detections in hpc environment. In *CCGrid 2005. IEEE International Symposium on Cluster Computing and the Grid, 2005.*, volume 1, pages 140–145. IEEE, 2005.

[34] Byung H Park, Saurabh Hukerikar, Ryan Adamson, and Christian Engelmann. Big data meets hpc log analytics: Scalable approach to understanding systems at extreme scale. In *2017 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 758–765. IEEE, 2017.

[35] Manish Kumar and M Hanumanthappa. Scalable intrusion detection systems log analysis using cloud computing infrastructure. In *2013 IEEE International Conference on Computational Intelligence and Computing Research*, pages 1–4. IEEE, 2013.

[36] Bingchang Liu, Wei Huo, Chao Zhang, Wenchao Li, Feng Li, Aihua Piao, and Wei Zou. αdiff: cross-version binary code similarity detection with dnn. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, pages 667–678, 2018.

[37] Lannan Luo, Jiang Ming, Dinghao Wu, Peng Liu, and Sencun Zhu. Semantics-based obfuscation-resilient binary code similarity comparison with applications to software and algorithm plagiarism detection. *IEEE Transactions on Software Engineering*, 43(12):1157–1177, 2017.

[38] Zeping Yu, Rui Cao, Qiyi Tang, Sen Nie, Junzhou Huang, and Shi Wu. Order matters: semantic-aware neural networks for binary code similarity detection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 1145–1152, 2020.

[39] LES Jaramillo. Malware detection and mitigation techniques: lessons learned from mirai ddos attack. *Journal of Information Systems Engineering & Management*, 3(3):19, 2018.

[40] Lingwei Chen, Yanfang Ye, and Thirimachos Bourlai. Adversarial machine learning in malware detection: Arms race between evasion attack and defense. In *2017 European Intelligence and Security Informatics Conference (EISIC)*, pages 99–106. IEEE, 2017.

[41] Gustavo Grieco, Guillermo Luis Grinblat, Lucas Uzal, Sanjay Rawat, Josselin Feist, and Laurent Mounier. Toward large-scale vulnerability discovery using machine learning. In *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy*, pages 85–96, 2016.

[42] Jannik Pewny, Behrad Garmany, Robert Gawlik, Christian Rossow, and Thorsten Holz. Cross-architecture bug search in binary executables. In *2015 IEEE Symposium on Security and Privacy*, pages 709–724. IEEE, 2015.

[43] Qian Feng, Rundong Zhou, Chengcheng Xu, Yao Cheng, Brian Testa, and Heng Yin. Scalable graph-based bug search for firmware images. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 480–491, 2016.

[44] Eui Chul Richard Shin, Dawn Song, and Reza Moazzezi. Recognizing functions in binaries with neural networks. In *24th {USENIX} Security Symposium ({USENIX} Security 15)*, pages 611–626, 2015.

[45] Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd international conference on Machine learning*, pages 369–376, 2006.

[46] John Lafferty, Andrew McCallum, and Fernando CN Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. 2001.

[47] Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. Siamese neural networks for one-shot image recognition. In *ICML deep learning workshop*, volume 2. Lille, 2015.

[48] Ahmed Saeed, Khaled A Harras, Ellen Zegura, and Mostafa Ammar. Local and low-cost white space detection. In *IEEE ICDCS*, 2017.

[49] Matt Ettus. USRP user's and developer's guide. *Ettus Research LLC*, 2005.

[50] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *ACM AsiaCCS*, 2017.

[51] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *IEEE S&P*, 2017.

[52] Yi Shi, Yalin Sagduyu, and Alexander Grushin. How to steal a machine learning classifier with deep learning. In *2017 IEEE International Symposium on Technologies for Homeland Security (HST)*, pages 1–5. IEEE, 2017.

[53] David H Wolpert and William G Macready. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82, 1997.

[54] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[55] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *IEEE EuroS&P*, 2016.

[56] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20, 1995.

[57] Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. Online passive-aggressive algorithms. *Journal of Machine Learning Research*, 7, 2006.

[58] Yalin E Sagduyu. Securing cognitive radio networks with dynamic trust against spectrum sensing data falsification. In *2014 IEEE Military Communications Conference*, pages 235–241. IEEE, 2014.

[59] Pramod K Varshney et al. Optimal data fusion in multiple sensor detection systems. *IEEE Transactions on Aerospace and Electronic Systems*, (1):98–101, 1986.

[60] Chunxiao Jiang, Yan Chen, KJ Ray Liu, and Yong Ren. Renewal-theoretical dynamic spectrum access in cognitive radio network with unknown primary behavior. *IEEE Journal on Selected Areas in Communications*, 31(3):406–416, 2013.

[61] Chunxiao Jiang, Yan Chen, Yang Gao, and KJ Ray Liu. Joint spectrum sensing and access evolutionary game in cognitive radio networks. *IEEE transactions on wireless communications*, 12(5):2470–2483, 2013.

[62] Yu Gan, Chunxiao Jiang, Norman C Beaulieu, Jian Wang, and Yong Ren. Secure collaborative spectrum sensing: A peer-prediction method. *IEEE Transactions on Communications*, 64(10):4283–4294, 2016.

[63] Zhengping Luo, Shangqing Zhao, Zhuo Lu, Jie Xu, and Yalin Sagduyu. When attackers meet ai: Learning-empowered attacks in cooperative spectrum sensing. *IEEE Transactions on Mobile Computing*, 2020.

[64] Report and order: In the matter of amendment of part 15 of the commission's rules for unlicensed operations in the television bands, repurposed 600 MHz band,600 MHz guard bands and duplex gap, and channel 37. *FCC ET Docket No. 14-165.*, 2015.

[65] VTCORNET.

[66] Fatemeh Amini and Mehdi Mahdavi. Local outlier factor based cooperation spectrum sensing scheme for defending against attacker. In *IEEE IST*, 2014.

[67] Peter J Rousseeuw and Katrien Van Driessen. A fast algorithm for the minimum covariance determinant estimator. *Technometrics*, 41, 1999.

[68] Ahmed Saeed, Khaled A Harras, Ellen Zegura, and Mostafa Ammar. Local and low-cost white space detection. In *IEEE ICDCS*, 2017.

[69] Ji Wang, Ray Chen, Jeffrey JP Tsai, and Ding-Chau Wang. Trust-based mechanism design for cooperative spectrum sensing in cognitive radio networks. *Computer Communications*, 116:90–100, 2018.

[70] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *IEEE CVPR*, 2016.

[71] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533*, 2016.

[72] Jonathan Uesato, Brendan O'Donoghue, Aaron van den Oord, and Pushmeet Kohli. Adversarial risk and the dangers of evaluating against weak attacks. *arXiv preprint arXiv:1802.05666*, 2018.

[73] Pin-Yu Chen, Yash Sharma, Huan Zhang, Jinfeng Yi, and Cho-Jui Hsieh. Ead: elastic-net attacks to deep neural networks via adversarial examples. *arXiv preprint arXiv:1709.04114*, 2017.

[74] Nicolas Papernot, Fartash Faghri, Nicholas Carlini, Ian Goodfellow, Reuben Feinman, Alexey Kurakin, Cihang Xie, Yash Sharma, Tom Brown, Aurko Roy, Alexander Matyasko, Vahid Behzadan, Karen Hambardzumyan, Zhishuai Zhang, Yi-Lin Juang, Zhi Li, Ryan Sheatsley, Abhibhav Garg, Jonathan Uesato, Willi Gierke, Yinpeng Dong, David Berthelot, Paul Hendricks, Jonas Rauber, and Rujun Long. Technical report on the cleverhans v2.1.0 adversarial examples library. *arXiv preprint arXiv:1610.00768*, 2018.

[75] Ankit Singh Rawat, Priyank Anand, Hao Chen, and Pramod K Varshney. Collaborative spectrum sensing in the presence of byzantine attacks in cognitive radio networks. *IEEE Transactions on Signal Processing*, 59(2):774–786, 2010.

[76] Federico Penna, Yifan Sun, Lara Dolecek, and Danijela Cabric. Detecting and counteracting statistical attacks in cooperative spectrum sensing. *IEEE Transactions on Signal Processing*, 60(4):1806–1822, 2011.

[77] Edward Peh and Ying-Chang Liang. Optimization for cooperative sensing in cognitive radio networks. In *IEEE WCNC*, 2007.

[78] Amir Ghasemi and Elvino S Sousa. Collaborative spectrum sensing for opportunistic access in fading environments. In *IEEE DySPAN*, 2005.

[79] Zhi Quan, Shuguang Cui, and Ali H Sayed. Optimal linear cooperation for spectrum sensing in cognitive radio networks. *IEEE J. Selected Topics in Signal Processing*, 2, 2008.

[80] Arash Ahmadfard, Ali Jamshidi, and Alireza Keshavarz-Haddad. Probabilistic spectrum sensing data falsification attack in cognitive radio networks. *IEEE Trans. Signal Process.*, 137, 2017.

[81] Zhengrui Qin, Qun Li, and George Hsieh. Defending against cooperative attacks in cooperative spectrum sensing. *IEEE Trans. Wireless Commun*, 12, 2013.

[82] Chowdhury Sayeed Hyder, Brendan Grebur, and Li Xiao. Defense against spectrum sensing data falsification attacks in cognitive radio networks. In *International Conference on Security and Privacy in Communication Systems*, pages 154–171. Springer, 2011.

[83] Lingjie Duan, Alexander W Min, Jianwei Huang, and Kang G Shin. Attack prevention for collaborative spectrum sensing in cognitive radio networks. *IEEE J. Selected Areas in Communications*, 30, 2012.

[84] Jamie Hayes and George Danezis. Machine learning as an adversarial service: Learning black-box adversarial examples. *arXiv preprint arXiv:1708.05207*, 2017.

[85] Yalin E. Sagduyu, Yi Shi, Tugba Erpek, Wiliam Headley, Bryse Flowers, George Stantchev, and Zhuo Lu. When wireless security meets machine learning: Motivation, challenges, and research directions. 2020. Available on arXiv:2001.08883.

[86] Yi Shi, Yalin E Sagduyu, Tugba Erpek, Kemal Davaslioglu, Zhuo Lu, and Jason H Li. Adversarial deep learning for cognitive radio security: Jamming attack and defense

strategies. In *2018 IEEE international conference on communications workshops (ICC Workshops)*, pages 1–6. IEEE, 2018.

[87] Tugba Erpek, Yalin E Sagduyu, and Yi Shi. Deep learning for launching and mitigating wireless jamming attacks. *IEEE Transactions on Cognitive Communications and Networking*, 5(1):2–14, 2018.

[88] Yi Shi, Tugba Erpek, Yalin E Sagduyu, and Jason H Li. Spectrum data poisoning with adversarial deep learning. In *MILCOM 2018-2018 IEEE Military Communications Conference (MILCOM)*, pages 407–412. IEEE, 2018.

[89] Yalin Sagduyu, Yi Shi, and Tugba Erpek. Adversarial deep learning for over-the-air spectrum poisoning attacks. *IEEE Transactions on Mobile Computing*, 2019.

[90] Zhengping Luo, Shangqing Zhao, Zhuo Lu, Yalin E Sagduyu, and Jie Xu. Adversarial machine learning based partial-model attack in iot. In *Proceedings of the 2nd ACM Workshop on Wireless Security and Machine Learning*, pages 13–18, 2020.

[91] Kemal Davaslioglu and Yalin E Sagduyu. Trojan attacks on wireless signal classification with adversarial machine learning. In *2019 IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN)*, pages 1–6. IEEE, 2019.

[92] Yi Shi, Kemal Davaslioglu, and Yalin E Sagduyu. Over-the-air membership inference attacks as privacy threats for deep learning-based wireless signal classifiers. In *Proceedings of the 2nd ACM Workshop on Wireless Security and Machine Learning*, pages 61–66, 2020.

[93] Yalin E Sagduyu, Yi Shi, and Tugba Erpek. Iot network security from the perspective of adversarial deep learning. In *2019 16th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*, pages 1–9. IEEE, 2019.

[94] Meysam Sadeghi and Erik G Larsson. Adversarial attacks on deep-learning based radio signal classification. *IEEE Wireless Communications Letters*, 8(1):213–216, 2018.

[95] Samuel Bair, Matthew DelVecchio, Bryse Flowers, Alan J Michaels, and William C Headley. On the limitations of targeted adversarial evasion attacks against deep learning enabled modulation recognition. In *Proceedings of the ACM Workshop on Wireless Security and Machine Learning*, pages 25–30, 2019.

[96] Silvija Kokalj-Filipovic, Rob Miller, and Garrett Vanhoy. Adversarial examples in rf deep learning: Detection and physical robustness. In *2019 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, pages 1–5. IEEE, 2019.

[97] Muhammad Zaid Hameed, András György, and Deniz Gündüz. Communication without interception: Defense against modulation detection. In *2019 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, pages 1–5. IEEE, 2019.

[98] Brian Kim, Yalin E Sagduyu, Kemal Davaslioglu, Tugba Erpek, and Sennur Ulukus. Over-the-air adversarial attacks on deep learning based modulation classifier over wireless channels. In *2020 54th Annual Conference on Information Sciences and Systems (CISS)*, pages 1–6. IEEE, 2020.

[99] Brian Kim, Yalin E Sagduyu, Kemal Davaslioglu, Tugba Erpek, and Sennur Ulukus. Channel-aware adversarial attacks against deep learning-based wireless signal classifiers. *arXiv preprint arXiv:2005.05321*, 2020.

[100] Brian Kim, Yalin E Sagduyu, Kemal Davaslioglu, Tugba Erpek, and Sennur Ulukus. How to make 5g communications" invisible": Adversarial machine learning for wireless privacy. *arXiv preprint arXiv:2005.07675*, 2020.

[101] Wenxiu Ding, Xuyang Jing, Zheng Yan, and Laurence T Yang. A survey on data fusion in internet of things: Towards secure and privacy-preserving fusion. *Information Fusion*, 51:129–144, 2019.

[102] Rohan Doshi, Noah Apthorpe, and Nick Feamster. Machine learning ddos detection for consumer internet of things devices. In *2018 IEEE Security and Privacy Workshops (SPW)*, pages 29–35. IEEE, 2018.

[103] Byungchul Tak, Seorin Park, and Prabhakar Kudva. Priolog: Mining important logs via temporal analysis and prioritization. *Sustainability*, 11(22):6306, 2019.

[104] Alina Sîrbu and Ozalp Babaoglu. A holistic approach to log data analysis in high-performance computing systems: The case of ibm blue gene/q. In *European Conference on Parallel Processing*, pages 631–643. Springer, 2015.

[105] Alessio Netti, Zeynep Kiziltan, Ozalp Babaoglu, Alina Sîrbu, Andrea Bartolini, and Andrea Borghesi. Online fault classification in hpc systems through machine learning. In *European Conference on Parallel Processing*, pages 3–16. Springer, 2019.

[106] Andrea Borghesi, Antonio Libri, Luca Benini, and Andrea Bartolini. Online anomaly detection in hpc systems. In *2019 IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pages 229–233. IEEE, 2019.

[107] Suraj P Kesavan, Takanori Fujiwara, Jianping Kelvin Li, Caitlin Ross, Misbah Mubarak, Christopher D Carothers, Robert B Ross, and Kwan-Liu Ma. A visual analytics framework for reviewing streaming performance data. *arXiv preprint arXiv:2001.09399*, 2020.

[108] FNU Shilpika, Bethany Lusch, Murali Emani, Venkatram Vishwanath, Michael E Papka, and Kwan-Liu Ma. Mela: A visual analytics tool for studying multifidelity hpc

system logs. In *2019 IEEE/ACM Industry/University Joint International Workshop on Data-center Automation, Analytics, and Control (DAAC)*, pages 13–18. IEEE, 2019.

[109] Ana Gainaru, Franck Cappello, Stefan Trausan-Matu, and Bill Kramer. Event log mining tool for large scale hpc systems. In *European Conference on Parallel Processing*, pages 52–64. Springer, 2011.

[110] Y-TW Wei-Yu Chen, Wen-Chieh Kuo, and Yao-Tsung Wang. Building ids log analysis system on novel grid computing architecture. *National Center for High-Performance Computing*, 2009.

[111] Devarshi Ghoshal and Beth Plale. Provenance from log files: a bigdata problem. In *Proceedings of the Joint EDBT/ICDT 2013 Workshops*, pages 290–297. ACM, 2013.

[112] Pinjia He, Jieming Zhu, Shilin He, Jian Li, and Michael R Lyu. An evaluation study on log parsing and its use in log mining. In *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 654–661. IEEE, 2016.

[113] Wei Hu, Yunrui Li, Vinay Srihari, and Ramana Yemeni. High-performance log-based processing, October 22 2013. US Patent 8,566,326.

[114] Richard S Sutton, Andrew G Barto, et al. *Introduction to reinforcement learning*, volume 2. MIT press Cambridge, 1998.

[115] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Thirtieth AAAI conference on artificial intelligence*, 2016.

[116] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. 2014.

[117] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

[118] Miguel Torres, R Vaughn, S Bridges, G Florez, and Z Liu. Attacking a high performance computer cluster. In *Proceedings of the 15th Annual Canadian Information Technology Security Symposium, Ottawa, Canada*, 2003.

[119] Prakashan Korambath. Cyber security in high-performance computing environment. *Institute for Digital Research and Education*, 2014.

[120] Farzad Sabahi. Cloud computing security threats and responses. In *2011 IEEE 3rd International Conference on Communication Software and Networks*, pages 245–249. IEEE, 2011.

[121] George Markowsky and Linda Markowsky. Survey of supercomputer cluster security issues. In *Security and Management*, pages 474–480, 2007.

[122] Bianca Schroeder and Garth A Gibson. A large-scale study of failures in high-performance computing systems. *IEEE transactions on Dependable and Secure Computing*, 7(4):337–350, 2009.

[123] Timothy W Curry et al. Profiling and tracing dynamic library usage via interposition. In *USENIX Summer*, pages 267–278, 1994.

[124] Jae-Kook Lee, Sung-Jun Kim, and Taeyoung Hong. Brute-force attacks analysis against ssh in hpc multi-user service environment. *Indian Journal of Science and Technology*, 9(24), 2016.

[125] Melisa Cantu, Joon Kim, and Xiaowen Zhang. Finding hash collisions using mpi on hpc clusters. In *2017 IEEE Long Island Systems, Applications and Technology Conference (LISAT)*, pages 1–6. IEEE, 2017.

[126] Brojo Kishore Mishra, Minakshi Sahu, and Satya Naryan Das. Intrusion detection systems for high performance computing environment. In *2014 International Conference on High Performance Computing and Applications (ICHPCA)*, pages 1–6. IEEE, 2014.

[127] Andrew Prout, William Arcand, David Bestor, Chansup Byun, Bill Bergeron, Matthew Hubbell, Jeremy Kepner, Peter Michaleas, Julie Mullen, Albert Reuther, et al. Scalable cryptographic authentication for high performance computing. In *2012 IEEE Conference on High Performance Extreme Computing*, pages 1–2. IEEE, 2012.

[128] Makan Pourzandi, David Gordon, William Yurcik, and Gregory A Koenig. Clusters and security: distributed security for distributed systems. In *CCGrid 2005. IEEE International Symposium on Cluster Computing and the Grid, 2005.*, volume 1, pages 96–104. IEEE, 2005.

[129] Ramya Prabhakar, Christina Patrick, and Mahmut Kandemir. Mpisec i/o: Providing data confidentiality in mpi-i/o. In *2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 388–395. IEEE, 2009.

[130] Byung H Park, Yawei Hui, Swen Boehm, Rizwan A Ashraf, Christopher Layton, and Christian Engelmann. A big data analytics framework for hpc log data: Three case studies using the titan supercomputer log. In *2018 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 571–579. IEEE, 2018.

[131] Adam Oliner and Jon Stearley. What supercomputers say: A study of five system logs. In *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*, pages 575–584. IEEE, 2007.

[132] Adam Oliner, Archana Ganapathi, and Wei Xu. Advances and challenges in log analysis. *Communications of the ACM*, 55(2):55–61, 2012.

[133] Edward Chuah, Shyh-hao Kuo, Paul Hiew, William-Chandra Tjhi, Gary Lee, John Hammond, Marek T Michalewicz, Terence Hung, and James C Browne. Diagnosing the root-causes of failures from cluster log files. In *2010 International Conference on High Performance Computing*, pages 1–10. IEEE, 2010.

[134] Liang Tang, Tao Li, and Chang-Shing Perng. Logsig: Generating system events from raw textual logs. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 785–794. ACM, 2011.

[135] Alexandru Iosup, Simon Ostermann, M Nezih Yigitbasi, Radu Prodan, Thomas Fahringer, and Dick Epema. Performance analysis of cloud computing services for many-tasks scientific computing. *IEEE Transactions on Parallel and Distributed systems*, 22(6):931–945, 2011.

[136] Marcello Cinque, Domenico Cotroneo, Roberto Natella, and Antonio Pecchia. Assessing and improving the effectiveness of logs for the analysis of software faults. In *2010 IEEE/IFIP International Conference on Dependable Systems & Networks (DSN)*, pages 457–466. IEEE, 2010.

[137] Siavash Ghiasvand, Florina M Ciorba, Ronny Tschüter, and Wolfgang E Nagel. Analysis of node failures in high performance computers based on system logs. In *28th ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC 2015)*. University_of_Basel, 2015.

[138] Nithin Nakka, Ankit Agrawal, and Alok Choudhary. Predicting node failure in high performance computing systems from failure and usage logs. In *2011 IEEE Interna-*

tional Symposium on Parallel and Distributed Processing Workshops and Phd Forum, pages 1557–1566. IEEE, 2011.

[139] Xiuqin Lin, Peng Wang, and Bin Wu. Log analysis in cloud computing environment with hadoop and spark. In *2013 5th IEEE International Conference on Broadband Network & Multimedia Technology*, pages 273–276. IEEE, 2013.

[140] Narate Taerat, Nichamon Naksinehaboon, Clayton Chandler, James Elliott, Chokchai Leangsuksun, George Ostrouchov, Stephen L Scott, and Christian Engelmann. Blue gene/l log analysis and time to interrupt estimation. In *2009 International Conference on Availability, Reliability and Security*, pages 173–180. IEEE, 2009.

[141] Min Du and Feifei Li. Spell: Streaming parsing of system event logs. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 859–864. IEEE, 2016.

[142] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael I Jordan. Detecting large-scale system problems by mining console logs. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pages 117–132. ACM, 2009.

[143] Qiang Fu, Jian-Guang Lou, Yi Wang, and Jiang Li. Execution anomaly detection in distributed systems through unstructured log analysis. In *2009 ninth IEEE international conference on data mining*, pages 149–158. IEEE, 2009.

[144] Adetokunbo AO Makanju, A Nur Zincir-Heywood, and Evangelos E Milios. Clustering event logs using iterative partitioning. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1255–1264. ACM, 2009.

[145] Ralf Reussner, Peter Sanders, and Jesper Larsson Träff. Skampi: A comprehensive benchmark for public benchmarking of mpi. *Scientific Programming*, 10(1):55–65, 2002.

[146] Shashank Tripathi, Siddhartha Chandra, Amit Agrawal, Ambrish Tyagi, James M Rehg, and Visesh Chari. Learning to generate synthetic data via compositing. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 461–470, 2019.

[147] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[148] Ralf C Staudemeyer and Eric Rothstein Morris. Understanding lstm–a tutorial into long short-term memory recurrent neural networks. *arXiv preprint arXiv:1909.09586*, 2019.

[149] Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[150] Alex Graves, Marcus Liwicki, Santiago Fernández, Roman Bertolami, Horst Bunke, and Jürgen Schmidhuber. A novel connectionist system for unconstrained handwriting recognition. *IEEE transactions on pattern analysis and machine intelligence*, 31(5):855–868, 2008.

[151] Sanket Shukla, Gaurav Kolhe, Sai Manoj PD, and Setareh Rafatirad. Stealthy malware detection using rnn-based automated localized feature extraction and classifier. In *2019 IEEE 31st international conference on tools with artificial intelligence (ICTAI)*, pages 590–597. IEEE, 2019.

[152] Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. Signature verification using a" siamese" time delay neural network. *Advances in neural information processing systems*, pages 737–737, 1994.

[153] Cheng Zhang, Wu Liu, Huadong Ma, and Huiyuan Fu. Siamese neural network based gait recognition for human identification. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2832–2836. IEEE, 2016.

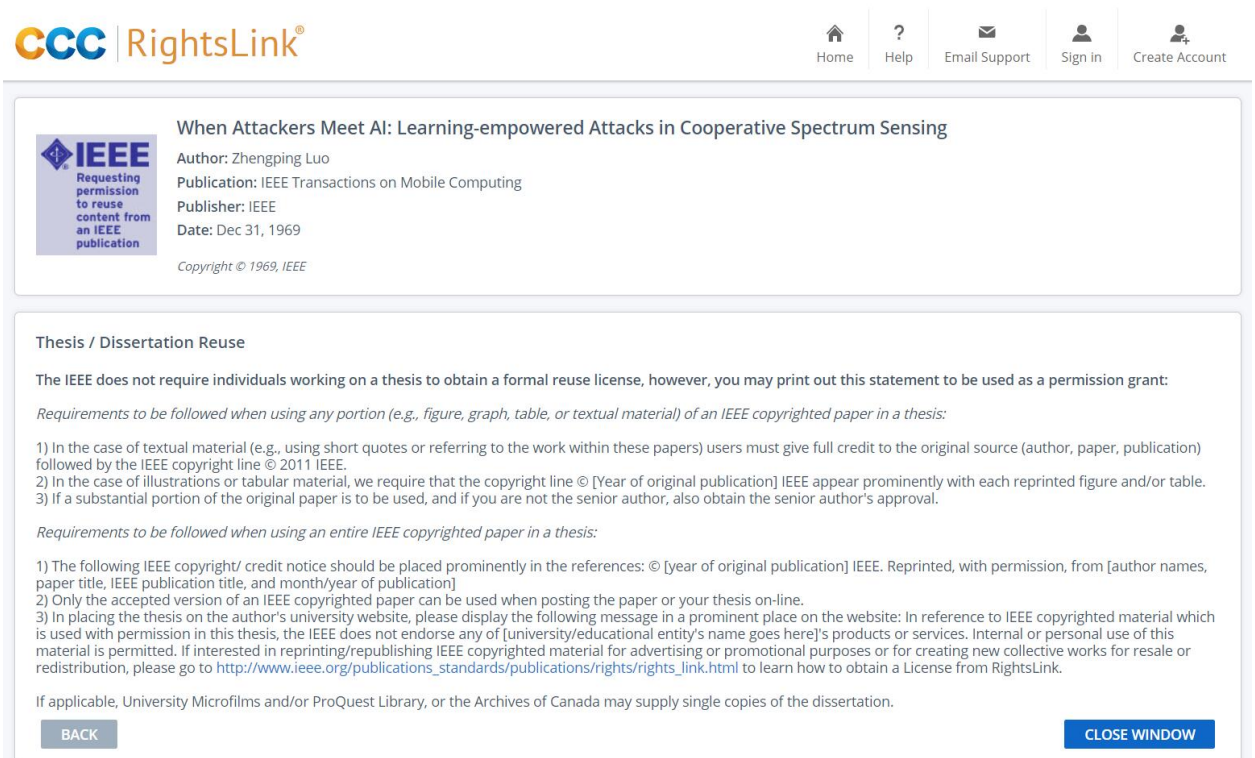
[154] Davide Chicco. Siamese neural networks: An overview. *Artificial Neural Networks*, pages 73–94, 2021.

[155] Samuel Berlemont, Grégoire Lefebvre, Stefan Duffner, and Christophe Garcia. Class-balanced siamese neural networks. *Neurocomputing*, 273:47–56, 2018.

[156] https://angr.io/.

Appendix A: Copyright Permissions

Bellow are permissions for the use of materials in Chapter 2,3,and 4.



sesa 19(21): e5

Research Article

Security of HPC Systems: From a Log-analyzing Perspective

Zhengping Luo[1,*], Zhe Qu[1], Tung Thanh Nguyen[2], Hui Zeng[2], Zhuo Lu[1]

1: Department of Electrical Engineering/Florida Center for Cybersecurity, University of South Florida, Tampa FL 33620, USA

2: Intelligent Automation Inc., Rockville MD 20855, USA

*Contact email: Zhengpingluo@mail.usf.edu

Abstract

High Performance Computing (HPC) systems mainly focused on how to improve performances of the computing. It has competitive processing capacity both in terms of calculation speed and available memory. HPC infrastructures are valuable computing resources that need to be carefully guarded and avoid being maliciously used. Thus, vulnerabilities are quintessential issues in HPC systems due to most of jobs and resources run or stored usually are sensitive and high-profit information. In this survey, we comprehensively review securities of HPC systems from a log-analyzing perspective, including well-known attacks and widely used defenses, especially intruder detection methods. We found that log files are used for the security purposes much less than what we expected. How to use all the available log files comprehensively and employ state-ofthe-art intrusion techniques to improve the robustness of HPC systems still lies for future research.

Keywords Security, high performance computing, attacks and defenses, intrusion detection, log file analysis

Received 2019-07-07 Accepted 2019-07-29 Published 2019-08-01 Publisher EAI

http://dx.doi.org/10.4108/eai.19-8-2019.163134