

University of Arkansas, Fayetteville  
**ScholarWorks@UARK**

---

Graduate Theses and Dissertations


---

12-2019

## Countering Cybersecurity Vulnerabilities in the Power System

Fengli Zhang  
*University of Arkansas, Fayetteville*

Follow this and additional works at: <https://scholarworks.uark.edu/etd>

 Part of the [Information Security Commons](#), [Software Engineering Commons](#), and the [Systems Architecture Commons](#)

---

### Citation

Zhang, F. (2019). Countering Cybersecurity Vulnerabilities in the Power System. *Graduate Theses and Dissertations* Retrieved from <https://scholarworks.uark.edu/etd/3556>

This Dissertation is brought to you for free and open access by ScholarWorks@UARK. It has been accepted for inclusion in Graduate Theses and Dissertations by an authorized administrator of ScholarWorks@UARK. For more information, please contact [scholar@uark.edu](mailto:scholar@uark.edu), [uarepos@uark.edu](mailto:uarepos@uark.edu).

Countering Cybersecurity Vulnerabilities in the Power System

A dissertation submitted in partial fulfillment  
of the requirements for the degree of  
Doctor of Philosophy in Computer Science

by

Fengli Zhang  
University of Science and Technology of China  
Bachelor of Engineering in Automation, 2014

December 2019  
University of Arkansas

This dissertation is approved for recommendation to the Graduate Council

---

Qinghua Li, Ph.D.  
Dissertation Advisor and Committee Chair

---

Xintao Wu, Ph.D.  
Committee Member

---

Brajendra Nath Panda, Ph.D.  
Committee Member

---

Roy A. McCann, Ph.D.  
Committee Member

## ABSTRACT

Security vulnerabilities in software pose an important threat to power grid security, which can be exploited by attackers if not properly addressed. Every month, many vulnerabilities are discovered and all the vulnerabilities must be remediated in a timely manner to reduce the chance of being exploited by attackers. In current practice, security operators have to manually analyze each vulnerability present in their assets and determine the remediation actions in a short time period, which involves a tremendous amount of human resources for electric utilities. To solve this problem, we propose a machine learning-based automation framework to automate vulnerability analysis and determine the remediation actions for electric utilities. Then the determined remediation actions will be applied to the system to remediate vulnerabilities. However, not all vulnerabilities can be remediated quickly due to limited resources and the remediation action applying order will significantly affect the system's risk level. Thus it is important to schedule which vulnerabilities should be remediated first. We will model this as a scheduling optimization problem to schedule the remediation action applying order to minimize the total risk by utilizing vulnerabilities' impact and their probabilities of being exploited.

Besides, an electric utility also needs to know whether vulnerabilities have already been exploited specifically in their own power system. If a vulnerability is exploited, it has to be addressed immediately. Thus, it is important to identify whether some vulnerabilities have been taken advantage of by attackers to launch attacks. Different vulnerabilities may require different identification methods. In this dissertation, we explore identifying exploited vulnerabilities by detecting and localizing false data injection attacks and give a case study in the Automatic Generation Control (AGC) system, which is a key control system to keep the power system's balance. However, malicious measurements can be injected to exploited devices to mislead AGC to make false power generation adjustment which will harm power

system operations. We propose Long Short Term Memory (LSTM) Neural Network-based methods and a Fourier Transform-based method to detect and localize such false data injection attacks. Detection and localization of such attacks could provide further information to better prioritize vulnerability remediation actions.

## ACKNOWLEDGMENTS

Here, I want to sincerely thank every one who has helped me through the journey to pursue my Ph.D. degree.

First of all, I want to give my huge thanks to my advisor Prof. Qinghua Li, who has given me a lot of help and direction through the past 5 years on my study and life. Working and collaborating with you is an enjoyable and unforgettable experience in my life. I appreciate all the time and patience you have spent on advising me. And thank for all your advice, idea, and contribution for my research and work. Your knowledge, dedication, and responsibility impressed me and influenced me all through the study journey.

Also, I want to express my gratefulness to all committee members, Prof. Xintao Wu, Prof. Roy A. McCann and Prof. Brajendra Nath Panda, who have helped me and given me valuable suggestions to my dissertation. I appreciate all your time, help and support on my Ph.D study and dissertation.

I also would like to give my thanks to my co-authors Philip Dale Huff and Kylie McClanahan. It is great experiences to work with you in the vulnerability and patch management project. Especially, I want to express my love and thanks to my beloved family and friends who have been supporting me all through my study journey.

This material is based upon work supported by the Department of Energy under Award Number DE-OE0000779. This work is also supported in part by NSF under award number 1751255.

## TABLE OF CONTENTS

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Automating Vulnerability Remediation Analysis . . . . .	1
1.2	Remediation Action Scheduling Optimization . . . . .	3
1.3	Identifying Exploited Vulnerabilities . . . . .	4
1.4	Organization . . . . .	7
<b>2</b>	<b>Automating Vulnerability Remediation Analysis</b>	<b>8</b>
2.1	Introduction . . . . .	8
2.2	Current Practice of VPM in Electric Utilities . . . . .	8
2.3	Machine Learning-based Framework for Remediation Action Analysis . . . . .	10
2.3.1	Vulnerability Features and Management . . . . .	12
2.3.2	Asset Features and Management . . . . .	13
2.3.3	Machine Learning Algorithm Selection . . . . .	15
2.3.4	Reason Code Generation . . . . .	17
2.3.5	Model Dynamic Training . . . . .	18
2.4	Instantiation of the Framework: A Case Study for an Electric Utility . . . . .	18
2.5	Evaluations . . . . .	20
2.5.1	Dataset . . . . .	20
2.5.2	Parameter Tuning for Decision Tree . . . . .	21
2.5.3	Prediction Accuracy . . . . .	22
2.5.4	Reason Code Verification . . . . .	24
2.5.5	Prediction with Dynamic Training . . . . .	26
2.5.6	Prediction with Different Feature Sets . . . . .	27
2.5.7	Prediction with Different Amounts of Training Samples . . . . .	28
2.5.8	Comparison with Other Models . . . . .	29
2.5.9	Remediation Analysis Delay and Cost . . . . .	30
2.6	Related Work . . . . .	31
2.7	Discussions . . . . .	33
2.8	Summary . . . . .	33
<b>3</b>	<b>Remediation Action Scheduling Optimization</b>	<b>34</b>
3.1	Introduction . . . . .	34
3.2	Approach . . . . .	34
3.2.1	Risk Metric . . . . .	35
3.2.2	Exploitability Prediction . . . . .	36
3.2.3	Patch Schedule Optimization . . . . .	38
3.3	Evaluation . . . . .	41
3.3.1	Evaluation on Exploitability Prediction . . . . .	41
3.3.2	Evaluation on Patch Scheduling . . . . .	43
3.4	Related Work . . . . .	47
3.5	Summary . . . . .	47

<b>4</b>	<b>Identifying Exploited Vulnerabilities through Data Forgery Attack Detection and Localization</b>	<b>49</b>
4.1	Introduction . . . . .	49
4.2	Preliminary . . . . .	51
4.2.1	AGC System . . . . .	51
4.2.2	Attack Models . . . . .	52
4.3	Detection Methods . . . . .	54
4.3.1	LSTM Neural Network . . . . .	54
4.3.2	LSTM-based Detection Method . . . . .	56
4.3.3	LSTM-based Attack localization . . . . .	58
4.3.4	Fourier Transform-based Method Detection and Localization . . . . .	59
4.4	Simulation System and Dataset . . . . .	60
4.4.1	Simulation System . . . . .	60
4.4.2	Simulation Dataset . . . . .	61
4.4.3	Real Dataset . . . . .	62
4.5	Evaluation . . . . .	62
4.5.1	Multi-Feature LSTM-based Method on Simulated Dataset . . . . .	63
4.5.2	Comparison with Single-Feature LSTM . . . . .	68
4.5.3	Fourier Transform-based Method on Simulated Dataset . . . . .	70
4.5.4	LSTM and Fourier Transform-based Methods on Real Dataset . . . . .	71
4.6	Related Work . . . . .	72
4.7	Summary . . . . .	73
<b>5</b>	<b>Conclusions and Future Work</b>	<b>74</b>
5.1	Conclusions . . . . .	74
5.2	Future Work . . . . .	75
	<b>Bibliography . . . . .</b>	<b>77</b>

## LIST OF TABLES

Table 2.1: Vulnerability Characteristics . . . . .	12
Table 2.2: Sample prediction results for three vulnerabilities . . . . .	20
Table 3.1: Extracted Features . . . . .	37
Table 3.2: Exploiting time prediction . . . . .	43



## LIST OF FIGURES

Figure 2.1:	Vulnerability and patch management process . . . . .	9
Figure 2.2:	Machine learning-based framework . . . . .	11
Figure 2.3:	An example of trained decision tree model . . . . .	16
Figure 2.4:	Prediction over min_leaf_samples . . . . .	21
Figure 2.5:	Prediction over tree depth . . . . .	21
Figure 2.6:	Prediction accuracy on 2017A and 2018A dataset . . . . .	23
Figure 2.7:	Time of reason code-based verification and feature-based verification	26
Figure 2.8:	Monthly prediction accuracy . . . . .	26
Figure 2.9:	Prediction on Different Feature Sets . . . . .	27
Figure 2.10:	Predictions over Different Training Sample Numbers . . . . .	28
Figure 2.11:	Comparison with other machine learning models . . . . .	29
Figure 3.1:	Distribution of vulnerabilities' exploited day after being published .	36
Figure 3.2:	Exploitability Prediction . . . . .	41
Figure 3.3:	Predicted probability of vulnerability $v_i$ . . . . .	43
Figure 3.4:	Predicted probability of vulnerability $v_k$ . . . . .	43
Figure 3.5:	Total amount of risks of unpatched vulnerabilities under one operator	44
Figure 3.6:	Number of vulnerabilities patched before exploited under one operator	44
Figure 3.7:	Time difference between patch time and exploit time if patched after exploited under one operator . . . . .	44
Figure 3.8:	Total amount of risks of unpatched vulnerabilities under multiple operators . . . . .	45
Figure 3.9:	Number of vulnerabilities patched before exploited under multiple operators . . . . .	46
Figure 3.10:	Time difference between patch time and exploit time if patched after exploited under multi operators . . . . .	46
Figure 4.1:	AGC System . . . . .	51
Figure 4.2:	Structure of Recurrent Neural Network . . . . .	54
Figure 4.3:	Structure of Long Short Term Memory Network . . . . .	55
Figure 4.4:	ACE data pattern . . . . .	57
Figure 4.5:	Prediction with LSTM . . . . .	57
Figure 4.6:	Moving Average of Normal ACE and ACE with Scale Attack . . . . .	59
Figure 4.7:	Fourier Transform of Moving Average with scale attacks . . . . .	59

Figure 4.8:	5-bus system with 2 control areas . . . . .	61
Figure 4.9:	ACE and ACE with Random Attack . . . . .	64
Figure 4.10:	Distances of Normal ACE and ACE with Random Attacks . . . . .	64
Figure 4.11:	Detection and Localization of Multi-Feature LSTM on Random Attack	65
Figure 4.12:	ACE and ACE with Ramp Attack . . . . .	65
Figure 4.13:	Distance of Normal ACE and ACE with Ramp Attacks . . . . .	65
Figure 4.14:	Detection and Localization of Multi-Feature LSTM on Ramp Attack	66
Figure 4.15:	Detection and Localization of Multi-Feature LSTM on Scale Attack	66
Figure 4.16:	ACE and ACE with Scale Attack . . . . .	66
Figure 4.17:	Distance of Normal ACE and Attacked ACE with Scale Attacks . .	66
Figure 4.18:	ACE and ACE with Min Attack . . . . .	67
Figure 4.19:	Distance of Normal ACE and Attacked ACE with Min Attacks . . .	67
Figure 4.20:	Detection and Localization of Multi-Feature LSTM on Min Attack .	67
Figure 4.21:	Detection and Localization of Multi-Feature LSTM on Max Attack .	67
Figure 4.22:	ACE and ACE with Max Attack . . . . .	68
Figure 4.23:	Distance of Normal ACE and Attacked ACE with Max Attacks . . .	68
Figure 4.24:	Comparison between Multi-feature LSTM and Single-feature LSTM on Random Attack . . . . .	69
Figure 4.25:	Comparison between Multi-feature LSTM and Single-feature LSTM on Ramp Attack . . . . .	69
Figure 4.26:	Comparison between Multi-feature LSTM and Single-feature LSTM on Scale Attack . . . . .	70
Figure 4.27:	Detection and Localization of Fourier Transform . . . . .	70
Figure 4.28:	Fourier Transform on Min Attack . . . . .	70
Figure 4.29:	Fourier Transform on Max Attack . . . . .	70
Figure 4.30:	Detection of Single-Feature LSTM and Fourier Transform on Random Attack . . . . .	71
Figure 4.31:	Detection of Single-Feature LSTM and Fourier Transform on Ramp Attack . . . . .	71
Figure 4.32:	Detection of Single-Feature LSTM and Fourier Transform on Scale Attack . . . . .	71

## List of Published Papers

1. **Chapter 2: Fengli Zhang**, and Qinghua Li, "Security Vulnerability and Patch Management in Electric Utilities: A Data-Driven Analysis," in *The 1st Radical and Experiential Security Workshop (RESEC)*, 2018.
2. **Chapter 4: Fengli Zhang** and Qinghua Li, "Deep Learning-Based Data Forgery Detection in Automatic Generation Control," in *IEEE Conference on Communications and Network Security (CNS): International Workshop on Cyber-Physical Systems Security (CPS-Sec)*, 2017.

## 1 Introduction

Vulnerabilities in software pose an important threat to power grid security since they could be exploited by adversaries to control power system computers and devices and launch devastating attacks. Even with a flawless security architecture, vulnerabilities are unavoidable and affect all installed software and hardware components. Moreover, electric utilities are moving toward a more interconnected grid, which carries the potential for higher risk exposure for those vulnerabilities. For those reasons, security vulnerability and patch management (VPM) is an integral and currently one of the most important components of power grid security [29].

Every month, many vulnerabilities are discovered [52]. In current VPM practice, electric utilities need to quickly analyze vulnerabilities, decide how to remediate the vulnerabilities and perform remediation actions to reduce security risks in the power systems. In this dissertation, we will study how to help counter these vulnerabilities and automate the VPM process. Specifically, we will automatically analyze the vulnerabilities and determine remediation actions, optimize remediation action applying order to reduce the system's security risk, and identify whether there is any vulnerability that has been exploited in this system to help better prioritize the remediation actions.

### 1.1 Automating Vulnerability Remediation Analysis

When security vulnerabilities with their assets are discovered, security operators in electric utilities need to decide how to remediate the vulnerabilities quickly to reduce security risks. However, making remediation decisions is not easy for electric utilities. First, although patching can fix vulnerabilities, it is not always possible or preferable to patch vulnerable assets since patching needs to reboot software and cause disruption of service.

Thus, although urgent patches are installed quickly, electric utilities usually install other patches to their assets under a certain maintenance schedule, e.g., quarterly. For some vulnerabilities that need timely attention but patching them might cause disruption of critical service, they can be mitigated first before being patched later in the next maintenance cycle. Second, vulnerabilities are far from equal in terms of security risk. As an example, the risk of a Google Chrome vulnerability applying to a supervisory control and data acquisition (SCADA) operator workstation with constant user browser activity differs drastically to an Internet browser vulnerability on an application server with no Internet access. An organization should immediately react to the former but can likely safely table the latter for a while. Remediation decisions should reflect the priority of vulnerabilities based on their risks to optimize the use of the limited security resources. Third, remediation decisions should consider many factors about vulnerabilities and assets, such as whether a vulnerability has exploit code available, whether the vulnerable asset is reachable from the Internet, the impact of exploits, and whether patching disrupts power delivery service, making it a complex reasoning process. Fourth, the volume of security vulnerabilities applicable to an electric utility at any given time often exceeds the capacity of organizations to apply risk analysis. Over the past few years, the National Institute of Standards and Technology (NIST) National Vulnerability Database (NVD) shows the number of software security vulnerabilities found and publicly reported has more than doubled [55]. It is not uncommon for an electric utility to have hundreds and even thousands of vulnerabilities each month for hundreds of or more assets.

Unfortunately, currently remediation decisions are manually made in electric utilities (at least in the U.S.). That induces long delays (typically in the order of weeks) in deciding the remediation actions for vulnerabilities. Such long latency delays the application of remediation actions and poses high security risks. The manual analysis also consumes a

tremendous amount of human time, which increases the cost of VPM.

To address this problem, we propose a machine learning-based framework to automate remediation decision analysis for electric utilities. The idea is to apply a predictive machine learning model over vulnerability features and asset features to predict the remediation decision for each vulnerability. The model can be built over historical, manual remediation decision data to capture and mimic how human operators make decisions, but it can make decisions much more quickly than manual analysis. Thus it has much shorter delays of remediation decision making which can reduce security risks while reducing the cost of VPM due to less manual efforts.

## 1.2 Remediation Action Scheduling Optimization

After we determine how to remediate the newly discovered vulnerabilities, the remediation actions should be performed to address the vulnerabilities. Since there are hundreds or even thousands of vulnerabilities to be remediated each month, not all vulnerabilities can be patched quickly due to limited resources. The order to remediate the vulnerabilities will heavily affect the security risk that the system is facing.

In practice, some organizations are based on operators' manual assessment or even their preference to apply remediation actions. Some organizations apply patches based on the prioritization order provided by the patch management software, where the patches are prioritized by vulnerabilities' impact such as Common Vulnerability Scoring System (CVSS) score [28]. These methods do not consider vulnerability exploitability dynamic attribute. They prioritize the order based on the vulnerability's attribute at the time point when the vulnerability is published. However, the exploitability of a vulnerability is changing over time and so is the risk that a vulnerability poses to a power system. As an example, if two vulnerabilities  $V_a$  and  $V_b$  are published at the same day.  $V_a$  with a lower CVSS score

could be exploited in 2 days after being published, and  $V_b$  with higher CVSS score might not be exploited even in 30 days after being published. If vulnerabilities are prioritized only based on CVSS score,  $V_b$  should be patched first. However, in this case, patching  $V_a$  first will be a better way to reduce the risk of being exploited. Thus it is important to schedule the remediation action applying order by considering the probabilities of being exploited to reduce the security risk.

We will schedule the remediation actions orders based on vulnerabilities' impact on the system and their probabilities of being exploited over time. One vulnerability's risk can be calculated with its impact on the system and its probability of being exploited. For many newly discovered vulnerabilities, since the probability of being exploited is unknown, we will first predict the probabilities of being exploited over time for these vulnerabilities. With the predicted probabilities, the vulnerabilities' security risk over time can be calculated. Then we can model this problem as an Mixed Integer Programming problem and schedule the remediation action order so that the total risk is as low as possible.

### 1.3 Identifying Exploited Vulnerabilities

Even though vulnerabilities' exploitability can be estimated, the electric utility still needs to know whether some vulnerabilities have already been exploited specifically for their power system. If a vulnerability is exploited, it has to be remediated immediately. Thus it is important to identify whether some vulnerabilities have been taken advantage of by attackers to launch attacks. Different vulnerabilities may require different identification methods. Here, we explore identifying exploited vulnerabilities by detecting and localizing false data injection attacks and give a case study in the Automatic Generation Control (AGC) system.

AGC is a key control system in the power grid which aims to keep balance between

power generation and load and maintain a stable power system frequency. It automatically adjusts power generation in response to area control imbalance. A power system usually consists of a number of interconnected control areas. A control area is connected to its neighboring areas through tie-lines, and power sharing between two neighboring areas occurs on these tie-lines. Each control area has its own AGC with the function to adjust the amount of power generation to keep its frequency in the scheduled range and keep the power exchanges with other areas to the values agreed upon in economic dispatch. The required adjustment in power generation in each control area is called Area Control Error (ACE), which is calculated based on the frequency deviation and power exchange deviation. It is calculated every 2-4 seconds and then the set-point of the generator governors participating in AGC is changed based on the calculated ACE.

In AGC, a control center periodically collects the power system's frequency and tie-line power flow measurements from meters to calculate ACE. If malicious tie-line power flow or frequency data is injected to normal measurement to mislead AGC to do miscalculations, AGC may issue wrong commands based on the miscalculated ACEs. As an example, when ACE is positive which implies the power is over generating, AGC should decrease the power generation. However, if the false tie-line power flow or frequency data injected results in a negative ACE value, AGC believes that the area is under generation and will issue a command to increase the power generation, which exaggerates the over generating situation.

False data can be injected to the system in two ways. One way is to directly inject the false data packets to the communication network channel. However, such attacks can be easily detected by authentication techniques. The other way is to compromise and control the devices such as meters to launch such attacks. If there are some unaddressed vulnerabilities existing in a device, attackers can exploit the vulnerabilities to compromise the device. The increasing number of smart devices in the power grid and the connection to external



networks make this way more feasible. If the false injected data are carefully designed, existing methods implemented in control centers such as State Estimation (SE) and Bad Data Detection (BDD) [18] are unable to detect intelligent cyber attacks. The work in [63] has shown that the existing schemes can be bypassed by carefully designed false data injection attacks.

If the attacks can be detected and localized, we will know which devices and vulnerabilities are exploited, and thus can better prioritize the vulnerability remediation. We propose Neural Network-based and Fourier Transform-based approaches to detect and localize false data injection attacks in AGC. In the first approach, we adopt Long Short Term Memory (LSTM) neural network to learn ACE patterns from historical data, predict ACE sequence pattern for next detection window based on the learned patterns, compare the predictions with the corresponding ACE sequence pattern calculated from measurements to determine whether there is forged data in the sequence. The LSTM model can be built with a single feature: the historical ACE data (single-feature LSTM) [65] [31]. It can also include more related features (i.e. multi-feature LSTM), such as frequency and tie-line power flow, to achieve better performance. The LSTM-based approach is also developed to localize attacks by checking which measurement is abnormal so that we can know which meter is compromised. In the second approach, we convert ACE and measurement data from the time domain to the frequency domain by using Fourier Transform and then check if ACE data is normal in the frequency domain [65]. We test these approaches on both simulated and real datasets. Since we only use historical data that are already available in current AGC systems, our detection methods can be easily deployed without interrupting normal services.

## 1.4 Organization

Chapter 2 discusses the work of vulnerability remediation analysis automation. Then we describe how to optimize the remediation action applying order in Chapter 3. In Chapter 4, we discuss a case study of AGC about exploited vulnerabilities identification. We conclude the dissertation work in the last chapter.

## 2 Automating Vulnerability Remediation Analysis

### 2.1 Introduction

In this chapter, we propose a machine learning-based framework to automate remediation decision analysis for electric utilities. The idea is to apply a predictive machine learning model over vulnerability features and asset features to predict the remediation decision for each vulnerability. The model can be built over historical, manual remediation decision data to capture and mimic how human operators make decisions. The machine learning model is also able to provide reasons why the predicted decisions are made.

The machine learning-based automation framework leverages two recent developments related to the electric sector. First, the North American Electric Reliability Corporation (NERC) Critical Infrastructure Protection (CIP) version 5 [8] regulatory requirements to maintain baseline configurations ensure the availability of well-formed software asset information in electric utilities. Second, the availability of well-formed and machine readable vulnerability information through the NIST NVD and third party service providers has significantly improved over the past few years [55].

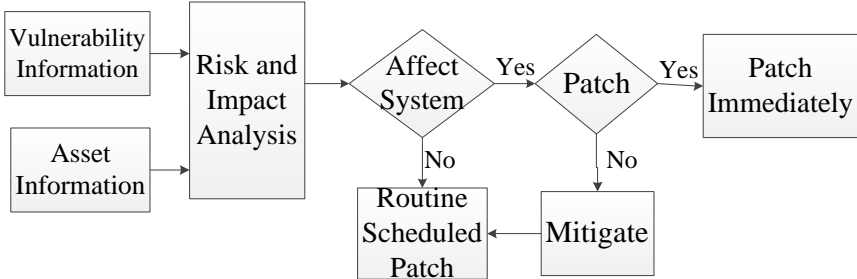
The chapter is organized as follows. Section 2.2 introduces current practices of VPM in electric utilities and presents results of a survey. Section 2.3 presents the machine learning-based automation framework. Section 2.4 introduces the instantiation of the framework in one electric utility. Section 2.5 presents evaluation results. Section 2.6 reviews related work. The last two sections present discussions and conclude the chapter.

### 2.2 Current Practice of VPM in Electric Utilities

As recommended practice for VPM by the U.S. Department of Homeland Security (DHS) [22], which is shown in Fig. 2.1, when vulnerabilities are discovered, organizations

first need to analyze whether a vulnerability will affect their systems by taking into consideration both vulnerability and asset information, and determine a remediation action for the vulnerability such as patching and mitigation.

It is not easy for utilities to perform VPM in practice. New vulnerabilities are discovered and new patches are released almost every day. Utilities have to spend a lot of time and human resources analyzing vulnerabilities and deciding on remediation actions. The NERC CIP standards [8] require strict monthly obligations for identifying and assessing security vulnerability and patches. Compliance to the standards is monitored closely through NERC and monetary penalties are regularly enforced. Likewise, the electric industry has a punitive incentive to closely follow these regulations.



**Figure 2.1:** Vulnerability and patch management process

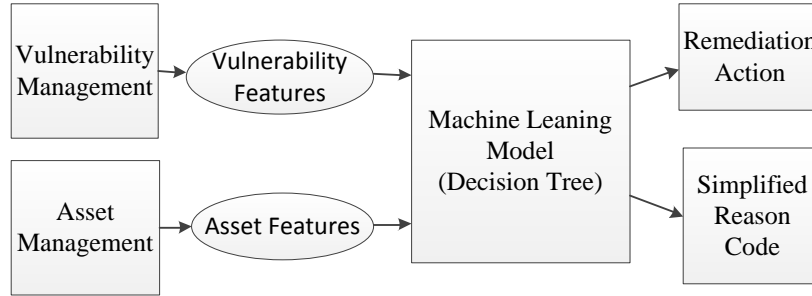
To gain more insights into the current practice, we also conducted a survey in the electric sector. The survey was distributed broadly to U.S. electric companies through national critical infrastructure protection groups and conducted anonymously due to constraints in sharing information so closely related to compliance with regulation. We received responses from 16 electric companies. 100% of the responded organizations perform manual analysis of vulnerabilities and patches. Around 60% of them need process more than 3000 security vulnerabilities each year and half of respondents spend more than 400 person hours monthly on VPM. All of them keep historical records of vulnerabilities and patches. The survey shows that VPM is indeed time-consuming and intensive work for utilities in practice.

### 2.3 Machine Learning-based Framework for Remediation Action Analysis

Security operators consider many factors to decide remediation actions for vulnerabilities. The factors include vulnerability information such as whether the vulnerability affects integrity, availability, or confidentiality, whether an exploit of the vulnerability is already available, what Common Vulnerability Scoring System (CVSS) score [28] is, and so on. The factors also include asset information such as whether the vulnerable device is a critical field device for power grid operations, whether the vulnerable device is sensitive to confidentiality/integrity/availability attacks, what the software is, and so on. Decisions are made considering the values of these factors. For example, if a vulnerability is at a non-critical device, only has little impact and there is no exploit available yet, it does not need to be addressed now and can be patched in the next scheduled cycle. If a vulnerability can be exploited, and it is at a user workstation, the decision is to patch immediately; if it is at a critical server, because patching a server may influence the power grid service, the decision is to mitigate it first and patch it later in the next scheduled cycle.

This process is tedious and repetitive, and we propose to automate remediation action analysis. Intuitively, one might consider manually making a set of rules (where each rule consists a combination of factor values for all factors and a decision for this combination) and use them to automate remediation action analysis similar to expert systems [37]. However, there are practical challenges with rule-based analysis: to cover all possible cases, the number of rules will grow exponentially. For example, at one of our utility partners, around 16 factors are considered and each factor has a number of values. The total number of possible combinations of factor values is about 240 billion. It is infeasible to manually generate so many rules in the first place, not to mention maintaining and updating them dynamically.

We adopt an approach that uses machine learning to automate the analysis. We



**Figure 2.2:** Machine learning-based framework

propose a machine learning-based framework (see Fig. 2.2) for remediation decision analysis which, based on vulnerability and asset features, automatically analyzes vulnerabilities and predicts remediation decisions, e.g., whether to patch them now or defer the patching to next regularly scheduled maintenance cycle. Central to the framework is the machine learning model, which not only outputs an remediation action but also an easy-to-verify reason code in case operators want to verify some predictions. The model can be trained from historical operation data that contains vulnerability information, asset information, and manual remediation decisions for a set of vulnerabilities. Our industry survey mentioned in Section 2.2 indicates that all the electric utilities surveyed maintain their historical operation data. This is expected specially in electric sectors because of the regulatory requirements for VPM. Our framework is consistent with the DHS guideline described in Section 2.2, but introduces machine learning-based automation to it and provides more details.

The goal of this work is to make machine learning predictions as accurate as manual decisions. That can help reduce security risks through making remediation decisions for vulnerabilities much more quickly and taking actions more quickly (see Section 2.5.9 for analysis).

**Table 2.1:** Vulnerability Characteristics

CVSS Score			User Interaction				Attack Vector			
Value in 0 - 10			High	Medium	Low		Network	Adjacent	Local	
Attack Complexity			Exploitability				Privilege			
High	Low		High	Functional	Proof-of-Concept	Unproven		High	Low	None
Confidentiality Impact			Integrity Impact			Availability Impact				
Complete	Partial	None	Complete	Partial	None		Complete	Partial	None	

### 2.3.1 Vulnerability Features and Management

Vulnerability features are already well established by the NVD. In the NVD, each vulnerability comes with a set of CVSS metrics that characterizes the vulnerability in different aspects. In our framework, we use CVSS metrics as vulnerability features since they are relevant to risk assessment and remediation decision analysis. The features and their possible values are shown in Table 2.1. The CVSS score is a number between 0 and 10 determined by the metrics to describe, in general, a vulnerability’s overall severity. Attack Vector shows how a vulnerability can be exploited, e.g., through the network or local access. Exploitability indicates the likelihood of a vulnerability being exploited. High as the highest level means exploit code has been widely available, and Unproven as the lowest level means no exploit code is available, with two other levels in between. User Interaction (Privilege, resp.) indicates the amount of user interaction (the level of privilege, resp.) needed by an attacker to exploit the vulnerability. The other four metrics describe the complexity of attack and the impact of attack in confidentiality, integrity, and availability. Detailed explanations of the metrics can be found in [28]. Each characteristic’s effect on the remediation decision making is also studied in the paper [66].

The NVD publishes vulnerabilities for a variety of software products daily. Each vulnerability is identified by a unique Common Vulnerabilities and Exposures (CVE) ID, such

as CVE-2016-8882. An organization can retrieve the vulnerabilities (including CVEs and vulnerability features) of their assets through identifying the Common Platform Enumeration (CPE) names [54] of their assets and then querying the NVD (NVD provides API for such queries) using the CPEs. CPE is a naming standard to represent software and structured in a manner making it possible to search applicable vulnerabilities [53] automatically. An organization can manually identify CPEs of their assets once and use them for years without needing to update them, and thus the maintenance cost is low.

It is worthy to note that the learning framework is general and can support other vulnerability features that a company might use (e.g., other features provided by third-party services). Also, if a company only uses part of the CVSS metrics in remediation decision analysis, then the learning model can be built upon those metrics.

### **2.3.2 Asset Features and Management**

Although vulnerability features have a well established CVE standard for maintaining publicly disclosed vulnerability information, vulnerability features alone do not provide sufficient information for meaningful remediation analysis on individual cyber assets. One treats very differently a vulnerable system providing direct services on the Internet (e.g. a web server) from the same vulnerability applying to an isolated system in a highly controlled local network. Likewise, browser vulnerabilities apply very differently to different cyber assets. Clearly an office computer primarily used for email and web browsing is significantly more vulnerable than a server with almost no user interaction which happens to have the same browser installed. The NVD CVSS system recommends to use three asset features, confidentiality requirement, integrity requirement, and availability requirement, to calculate environment scores. However, only the three asset features are insufficient and security operators not only consider these features when deciding vulnerability remediation. For example,



whether the asset can be accessed externally is a critical factor when a vulnerability can be launched through network. Thus we identify two more asset features to complement those three. Another difference from the environmental CVSS system is that our approach uses machine learning to integrate these features into one decision making model rather than calculating environmental scores using simple formulas. All the five asset features are described below.

- Workstation User Login: (Yes or No) - Associates with the vulnerability user interaction feature. Whether the cyber asset provides an interactive workstation for a human operator. If the cyber asset does not have interactive use, then vulnerabilities affecting applications such as web browsers would have significantly less impact.
- External Accessibility: (High, Authenticated-Only or Limited) - Associates with the vulnerability attack vector feature. The degree to which cyber assets are externally accessible outside of the cyber system. For example, High may mean a web server providing public content, and Authenticated-Only may be a group of remotely accessible application servers which require login before use. Limit means there is no direct connectivity to the external network (but it could be connected to a device that is externally accessible).
- Confidentiality Requirement: (High, Medium or Low) - Associated with the vulnerability confidentiality impact feature. If the confidentiality requirement of the asset is set as “High”, loss of confidentiality will have severe impact on the asset.
- Integrity Requirement: (High, Medium or Low) - Similar to Confidentiality Requirement but focused on integrity.
- Availability Requirement: (High, Medium or Low) - Similar to Confidentiality Re-

quirement but focused on integrity.

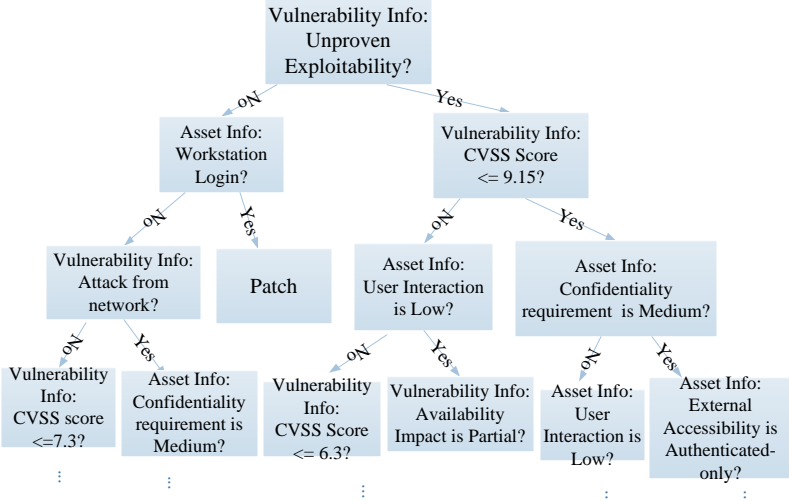
Again, an organization can always customize for their needs by adding and/or removing some asset features based on their operation practice.

**Group-based asset feature management** Whereas the software vulnerability feature set has a worldwide community to maintain consistent machine-readable features (i.e. through the NVD), the cyber asset features must be maintained by the organization. According to our survey, one organization can have thousands of assets and assets may change frequently (i.e. one asset may be removed or another new asset might be added). Due to the large amount of assets and their instability, it is cumbersome to analyze and maintain the characteristic values for each asset. In order to reduce the cost of maintenance, we divide assets into asset groups based on their roles or functions. For example, all Remote Terminal Units (RTUs) of a specific vendor and function can be categorized into one group since they have similar features. Similarly, all firewalls can be in one group. The assets in the same group share the same set of values for asset features. Then human operators can determine and maintain the feature values for each group. In our experiments with the electric utilities, categorization groups remained mostly consistent through large increases in asset population. For example, an operator workstation in a control center has the same features whether there are five or 100. Although assets may come and go, we find it is much less common for an entirely new asset group to appear. Since the number of groups is much smaller than the number of assets, grouping will greatly reduce the amount of efforts needed in maintaining feature values.

### 2.3.3 Machine Learning Algorithm Selection

Many machine learning algorithms are available today and we need to identify the best one to solve our problem. In this framework, we adopt the decision tree model to automate

remediation action analysis for the following reasons: (1) Decision tree-based decision making resembles human reasoning. On each level of the tree, the model chooses the most important factor and splits the problem space into multiple branches based on the factor’s value. Unlike many other machine learning models such as neural networks and logistic regression that are not very transparent, the decision tree model allows us to see what the model does in every step and know how the model makes decisions. Thus the predictions from decision tree can be interpreted, and a reason code can be derived to explain predictions. Human operators can verify the predictions based on reason code. (2) For this VPM automation problem, decision tree is proven to have very good performance in our experiments compared with several other machine learning algorithms.



**Figure 2.3:** An example of trained decision tree model

For illustration purposes, Fig. 2.3 shows a sample trained decision tree model in the remediation action analysis context. The prediction process for a vulnerability based on this tree is as follows. When a new vulnerability data record is fed into the model for prediction, the model will first look at the exploitability feature at the root node. If the exploitability is not Unproven, it will go to check the asset feature ”workstation login”. If the workstation allows user login, it means it faces more risks and must be patched immediately. Other tree

branches can be traversed by other records similarly.

### 2.3.4 Reason Code Generation

It is very difficult for a predictive machine learning tool to be 100% accurate. To increase transparency in the predictive decision, our approach provides human operators with both prediction confidence and a readable description (called reason code) of why the model selected the decision. The use of decision tree makes reason code generation feasible. A trained decision tree model is a collection of connected nodes and splitting rules organized in a tree structure. Then the reason code for each leaf node (decision node) can be derived by traversing the tree path and combining the splitting rules of the nodes in the path. However, for some long paths (e.g., one tree we built over a utility's data has an 18-node path), the reason code could become very long, redundant, and hard to read if simply combining the splitting rules. To address this issue, we use two rules to simplify and shorten the raw reason codes derived from the decision path.

- Intersection: we can reduce redundancy by finding range intersection. For example, for continuous data such as CVSS scores, if one condition in the reason code is “CVSS Score is larger than 5.0” and the other condition is “CVSS Score is larger than 7.0”, then we can find the intersection and the reason code can be reduced to “CVSS Score is larger than 7.0”.
- Complement: for some features that appear in several conditions of a path, we can replace these conditions by using its complementary condition. For example, for integrity impact, the set of possible values is Complete, Partial, None. If the reason code is “Integrity impact is not None, and Integrity impact is not Partial”, since the complement of Partial, None is Complete, the reason code can be reduced to “Integrity is Complete”.

### 2.3.5 Model Dynamic Training

The decision rationales of an electric utility are usually quite stable, but there are still changes in a longer time scale. Thus, the decision tree model should be dynamically updated and trained with the most recent historical data to capture the changes. For example, in each month (which is the typical working cycle of VPM in electric utilities), the machine learning model is retrained with the previous  $n$  ( $n$  can be customized for each organization) months' data. Note that the predicted decisions output by the model cannot be used as training data since the predictions are not always accurate. To address this issue, for each month's predicted decisions, human operators can verify a small portion of the predictions (e.g., 10%) to check the framework's performance, and these manually verified/checked vulnerabilities and predictions can be used as training data for retraining the model. Our experiments will show that a small portion of manual verification each month will achieve high prediction accuracy for the retrained model (see Section 2.5).

## 2.4 Instantiation of the Framework: A Case Study for an Electric Utility

To study how the machine learning-based framework works in the real world, we apply and instantiate the framework for one electric utility based on its VPM operation practices and data. Due to the high sensitivity of the VPM operation information, the company required us to anonymize its name, and thus we refer to it as  $Org_A$  in this paper.

In this instance of the framework, the vulnerability features used are the nine attributes in CVSS metrics, i.e., CVSS Score, Exploitability, Attack Vector, Attack Complexity, User Interaction, Privilege, Confidentiality Impact, Integrity Impact, and Availability Impact. The asset features used are Workstation User Login, External Accessibility, Confidentiality Requirement, Integrity Requirement, and Availability Requirement. These fea-

tures are used by human operators when they make remediation decisions. The possible remediation actions are Patch-Now, Mitigate-Now-Patch-Later, and Patch-Later, which are also used by the operators.

To get asset features, an asset list is first obtained from the company's baseline configuration management tool. Then the assets are grouped into 43 groups based on their functions. For each asset group, the value for each asset feature is identified. For the company, these asset features and feature values are quite stable, with no need to change in years. To get vulnerabilities and vulnerability features, a CPE is generated for each software/firmware based on software/firmware name and version which are also available in the baseline configuration management tool. Then we use a Python program developed by us to automatically query the NVD through its API using the CPE as a parameter and retrieve applicable vulnerabilities including their CVEs and CVSS vectors from the NVD. Vulnerability retrieval needs to be done pretty frequently, and the Python program makes it automatic and easy. Note that third party services that aggregate vulnerabilities for utilities also provide the same CVE and CVSS information usable in our framework.

The decision tree is implemented in Python based on the library Scikit-learn. The utility maintains VPM operation data including historical vulnerabilities, their associated assets, and the manual remediation decisions for them made by operators. That allows a decision tree model to be trained using the utility's historical data. It is worthy to note that some decision tree parameters should be tuned based on the dataset to achieve best performance (see Section 2.5.2 for details). After the model is trained, when a new vulnerability is obtained, its vulnerability features together with its asset features will be fed into the decision tree model for analysis.

The model outputs three pieces of information: predicted decision, confidence, and reason code. The confidence valued between 0 and 1 shows how confident the model is

**Table 2.2:** Sample prediction results for three vulnerabilities

Predicted action	Confidence	Reason code
Patch-Later	1	Unproven Exploitability, CVSS Score is less than 4.2 and Medium Confidentiality Impact
Mitigate-Now-Patch-Later	0.91	Proof-of-Concept Exploitability, Network Attack, High External Accessibility and High Confidentiality Impact
Patch-Now	1	not Unproven Exploitability and this Workstation allows users' login

with the prediction. It can guide operators to select predictions for manual verification, e.g., verifying those predictions with low confidence. Reason code helps human operators to understand and verify the prediction. Table 2.2 shows examples of the predictions for three different vulnerabilities. The first one shows that the predicted action is ‘Patch Later’ with 100% confidence. The reasoning for that choice is that the vulnerability is not exploitable, the CVSS score is less than 4.2, which suggests a low asset impact, and it has a medium confidentiality impact. The other two predictions can be interpreted in a similar way. We will present detailed experiment results in the next section.

## 2.5 Evaluations

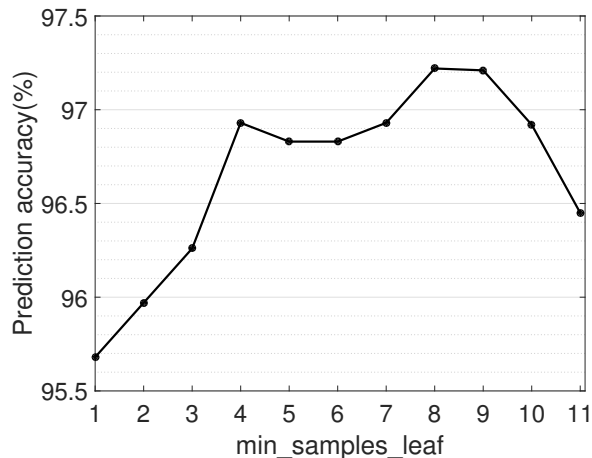
This section presents experimental results for the instance of the framework described in Section 2.4.

### 2.5.1 Dataset

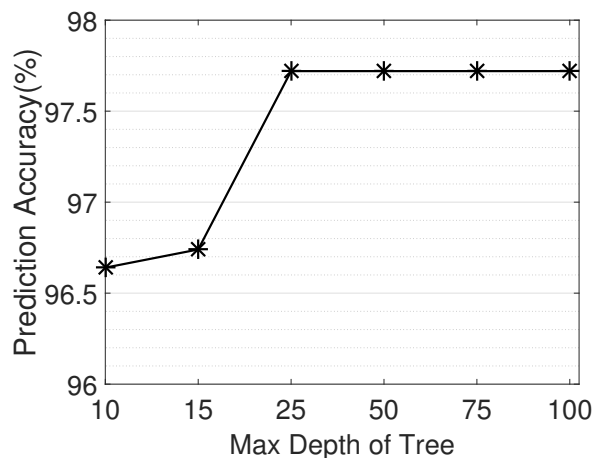
We collected two datasets from *Org<sub>A</sub>*, each containing one year of data. One dataset was collected from June 2016 to May 2017, with 3,476 vulnerability data records. The other dataset was collected from January 2018 to December 2018, with 3,660 records. For convenience, we refer to the two datasets as *2017A* and *2018A* respectively. Each vulnerability data record includes the following information: its associated software, vulnerability fea-

tures, its associated asset, and asset features. Additionally, each record also includes the remediation decision for the vulnerability made by human operators.

### 2.5.2 Parameter Tuning for Decision Tree



**Figure 2.4:** Prediction over min\_leaf\_samples



**Figure 2.5:** Prediction over tree depth

To prevent the tree from going too deep and avoid overfitting, the minimum number of samples at a leaf node (if the number of samples in a node is no more than the minimum number, the node will stop splitting) and the maximum depth of the tree should be properly set. These two parameters can be tuned based on the deployment environment. In our implementation, the 2017A dataset from the utility is used to tune these two parameters. In particular, a random 70% of the dataset was used as training data and the other 30% was used as testing data, and the two parameters were tuned based on the testing performance.

We experimented on different minimum numbers of samples in leaf nodes and the results are shown in Fig. 2.4. “min\_samples\_leaf” is the minimum number of samples required for a leaf node. The smaller “min\_samples\_leaf” is, the more the tree splits and the deeper and larger the tree is. As shown in Fig. 2.4, when “min\_samples\_leaf” is 8, it has highest prediction accuracy 97.22%. Here, Prediction accuracy is defined as the fraction of predicted



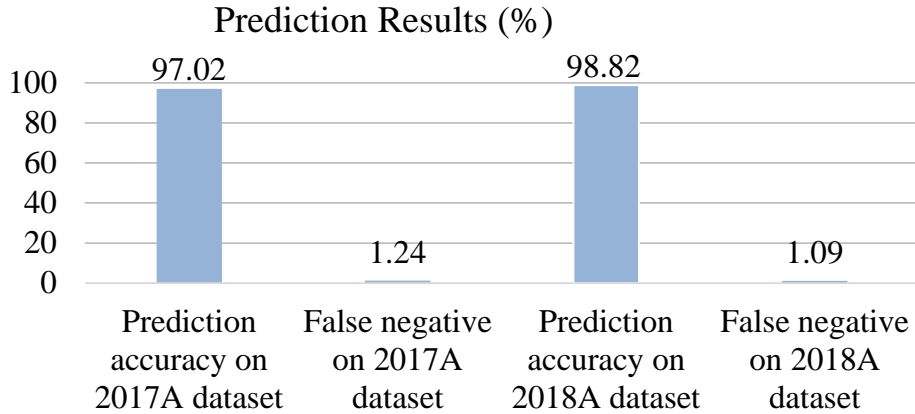
decisions that are the same as human operator’s manual decisions. When “min\_samples\_leaf” decreases, the prediction accuracy decreases since the tree is too specific to generalize to new samples. If “min\_samples\_leaf” is too large where the tree is short and small, the prediction accuracy also decreases because the trained tree does not capture sufficiently fine-grained information of the training data. Thus we set the minimum number of samples in leaf nodes as 8 for the remaining experiments.

The prediction accuracy under different tree max depth is shown in Fig. 2.5. When the tree depth goes over 25, the prediction accuracy does not change any more. Usually, when the max depth is larger, the tree is allowed to go deeper and there will be an overfitting issue. However, in this situation, since the the minimum number of leaf is set as 8, the tree will stop splitting when the leaf samples is equal to or less than 8 and thus it cannot go too deep. We set the tree max depth as 50 for the remaining experiments.

### 2.5.3 Prediction Accuracy

In this experiment, we test the model over the two datasets from organization  $Org_A$  separately. Each dataset is randomly split into two parts, 70% for training and 30% for testing. We use prediction accuracy and false negative rate to describe the performance. The false negative rate is defined as the fraction of predictions where the manual decision is Patch-Now or Mitigate-Now-Patch-Later but the prediction is Patch Later. False negative rate should be minimized since it delays the remediation of vulnerabilities while they should be addressed more timely. The results are shown in Fig. 2.6. For the 2017A dataset, the prediction accuracy is 97.02%, and false negative is 1.24%. For the 2018A dataset, the prediction accuracy is 98.82% and the false negative is 1.09%. The accuracy is quite high, which means using machine learning to predict remediation actions is feasible.

**Exploration of False Prediction** Although the accuracy of machine learning pre-



**Figure 2.6:** Prediction accuracy on 2017A and 2018A dataset

diction is already high, we still want to figure out what caused false predictions. For the 2017A dataset, after exploring the 2.98% of falsely predictions, we found that they were mainly caused by inconsistent manual remediation decisions in the historical dataset where some vulnerabilities with identical features were given different manual decisions, which can confuse the decision tree (and actually any other learning algorithm). The same problem caused the prediction error over the 2018A dataset as well.

This situation happens for several reasons. In a company, there are usually a group of security operators analyzing vulnerabilities and making remediation decisions. Different operators may have different decisions for vulnerabilities with identical features and even for the same vulnerabilities. Even if there is only one operator, for two vulnerabilities with identical features, s/he might make different decisions when processing them at different times (e.g., last week and this week). This is especially possible for vulnerabilities whose risk level is not at the clearly high risk end (which typically goes to Patch-Now and Mitigate-Now-Patch-Later) or the clearly low risk end (which typically goes to Patch Later) but goes in the middle. This is a kind of human mistake that operators cannot totally avoid.

For each set of vulnerability records that have identical features but different manual decisions, if we assume the majority decision of this set is the correct decision and the

records with minority decisions are deemed errors and removed from the dataset (about 3% records are removed for the 2017A dataset), then the prediction accuracy achieves 99.8% and the false negative rate achieves 0.20% for the 2017A dataset, with similar improvement for the 2018A dataset. This result shows that the prediction performance can be improved significantly if there are less inconsistent manual decisions in the training dataset. We plan to further explore this problem in collaboration with the utility company and improve our design in future work.

We also found that, for records with same features but different manual decisions, their prediction confidence will be relatively low. For example, suppose one leaf node of the decision tree contains four records with exactly the same features, and three of them were remediated by Patch-Now and one by Mitigate-Now-Patch-Later. Then the decision tree will output Patch-Now as the predicted decision with confidence 0.75. If operators are able to verify/correct the predictions with relatively low confidence (i.e. under 0.9), that can improve the performance to 99.42% accuracy and 0.38% false negative for the 2017A dataset and 99.45% accuracy and 0.09% false negative. Since there are only about 10% of predictions with confidence under 0.9, the manual verification time will be much shorter than manually making all the remediation decisions.

#### **2.5.4 Reason Code Verification**

Each prediction comes with one reason code so that users can verify the prediction when needed. Here, we first look into the length of reason code. For reason codes, we use the number of conditions that a reason code has to denote its length. For example, the length of reason code “Unproven Exploitability, CVSS Score is less than 4.2, and Medium Confidentiality Impact” is 3 because it includes 3 conditions. For predictions described in Section 2.5.3, the average length of reason code is 6.9 conditions. After applying the

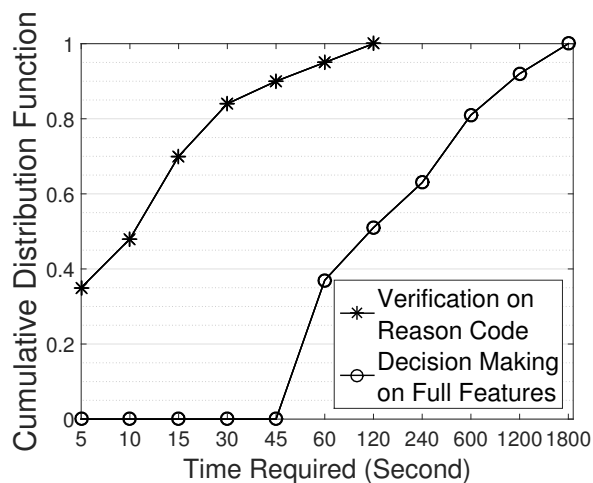
length reduction rules proposed in Section 2.3.4, the average length is reduced to 3.6 conditions. For example, the reason code “Unproven Exploitability, CVSS Score is less than 9.15, External Accessibility is not High, CVSS Score is less than 6.30, External Accessibility is not Authenticated-Only, and Medium Availability Impact” can be reduced to “Unproven Exploitability, CVSS Score is less than 6.3, Limited External Accessibility, and Medium Availability Impact”. The length reduction rules can significantly reduce the length of the reason codes so that it can be easier to understand and verify.

We then test whether the reason codes generated by the tool are sufficient to verify predicted decisions, check the time needed to verify the reason codes, and compare it with the time needed to verify predictions based on the corresponding vulnerabilities’ raw features. To do this, we randomly selected 100 predictions from the testing data, and asked a security operator from the organization  $Org_A$  to verify these predictions based on reason codes and based on the raw features.

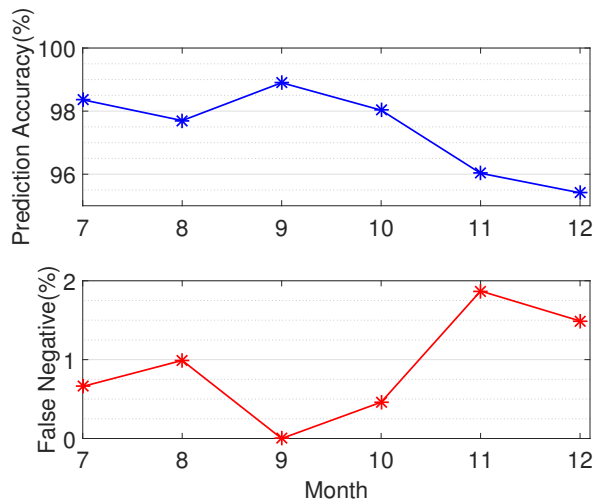
Results show that 98 out of the 100 reason codes are sufficient to verify the predicted decisions. One prediction is found to be wrong through the reason code verification. The other one reason code is insufficient to verify the prediction.

The time spent on reason code verification and raw feature based verification is shown in Fig. 2.7. Most of the reason codes can be verified in a very short time. 35% of reason codes can be verified in 5 seconds, and 90% in 45 seconds. The average verification time is 28.8 seconds. From the figure, it can be seen that verification based on raw features requires much more time than verification based on reason code. Only 35% of predictions can be verified in 60 seconds and about 40% takes over 4 minutes to verify. The average time of raw feature based verification is 7 minutes. The results show that the efficiency of reason codes. It is reasonable, because reason codes are derived (and optimized) from the decision tree and the decision tree to some extent prioritizes the judging conditions in the decision making

process based on their importance and hides unimportant factors from being considered.



**Figure 2.7:** Time of reason code-based verification and feature-based verification



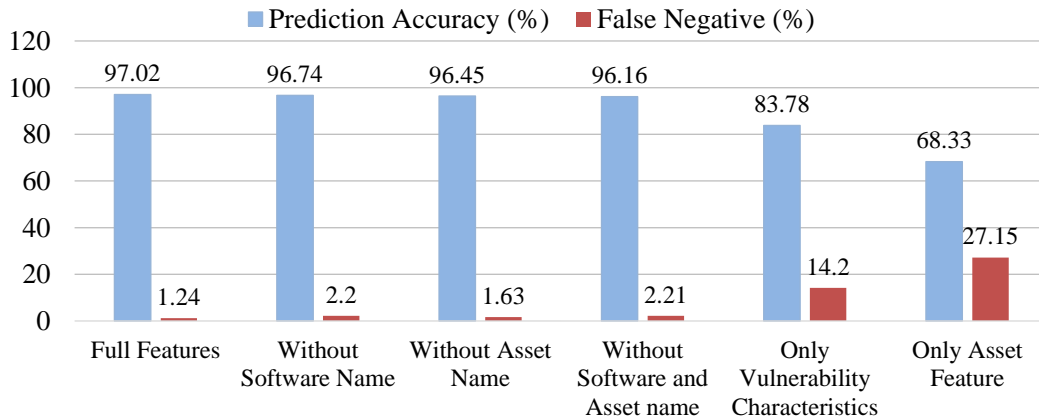
**Figure 2.8:** Monthly prediction accuracy

### 2.5.5 Prediction with Dynamic Training

In the above experiments, the twelve months’ data are randomly split into training data and testing data. In practice, the decision tree model should be dynamically updated and trained with recent historical data as discussed in Section 2.3.5. In this experiment, we test the model’s prediction accuracy with dynamic training. In particular, we assume the operator randomly selects 10% of each month’s predictions to check and verify. At each month, we use the recent six months’ vulnerabilities and decisions that have been manually checked by operators as training data to train a new decision tree model, and use it to predict for next month. The prediction results over the 2018A dataset are shown in Fig. 2.8. The x-axis means which month it is predicted for and y-axis is the prediction accuracy. For example, when the x-axis is 7, it uses 10% of the first six months’ data to train the model and predicts decisions for month 7. Then it uses 10% of the data from the second month to the seventh month to train the model and predict for the eighth month’s vulnerabilities. It can be seen that the prediction accuracy is not the same for different months, but overall

it is high. The prediction performance for the 2017A dataset under dynamic training has similar trends. Thus, dynamic training is feasible.

### 2.5.6 Prediction with Different Feature Sets



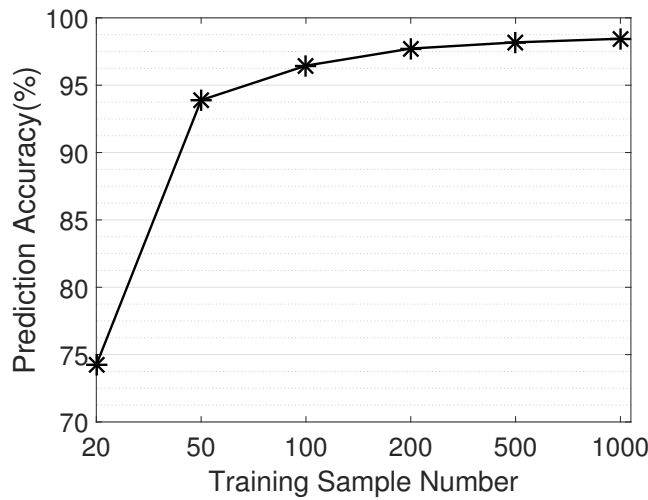
**Figure 2.9:** Prediction on Different Feature Sets

In general, the more useful features are used, the better the machine learning performs. In the above testing experiments, we use 16 features consisting of software name, vulnerability characteristics, asset name, and asset features. In these features, vulnerability characteristics and asset characteristics are stable since their set of possible feature values (e.g., 0-10 for CVSS score) does not change. However, the software name and asset name have infinite possible values and are very easy to change. For example, there might be a totally new software name that the decision tree model has never seen before. Such changeable features might induce more effort to maintain the machine learning model. Thus, we test how the machine learning model performs software name or asset name. We also explore how the model performs only with the vulnerability characteristics, and only with asset characteristics.

The prediction performances on different feature sets over the 2017A dataset are shown in Fig. 2.9. The figure shows the prediction performance when using the features

without software name, features without asset name, features without software name and asset name, only vulnerability features, and only asset features respectively. We can see that without software name or asset name as features, the prediction accuracy decreases about 0.3%, which is still good performance. However, with only vulnerability characteristics as features, the prediction accuracy is about 83.78%, and with only asset characteristics as features the prediction accuracy is only 68.33%. The results indicate that both vulnerability and asset characteristics should be considered as features in the machine learning model.

### 2.5.7 Prediction with Different Amounts of Training Samples

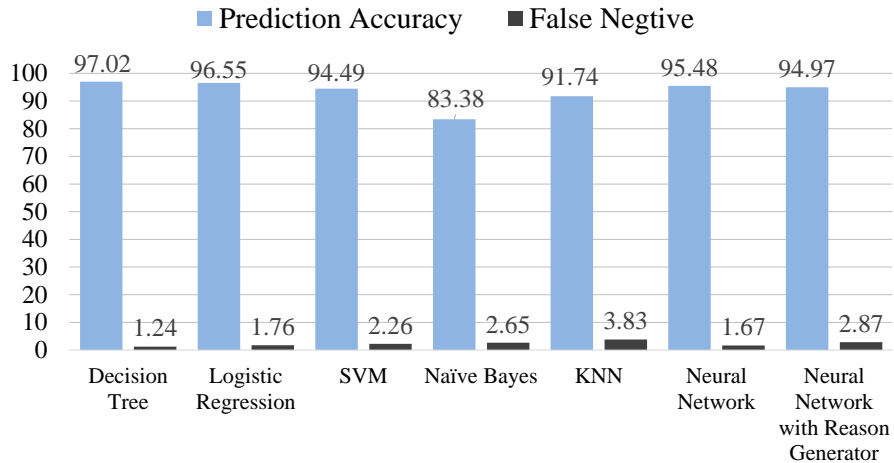


**Figure 2.10:** Predictions over Different Training Sample Numbers

Training data is required to enable the machine learning model to work. Some organizations may not have their historical vulnerability and remediation decision data. One way to generate training data is to let security operators manually make remediation decisions for some vulnerabilities. Here we explore how many training data samples are needed so that the model has good prediction performance. The prediction accuracy over different training sample numbers of 2018A dataset is shown in Fig. 2.10. Generally, the larger the training dataset is, the higher the prediction accuracy is. When there are 100 training samples, the

prediction accuracy is over 95%, which means the learning model is easy to build up for a reasonable performance.

### 2.5.8 Comparison with Other Models



**Figure 2.11:** Comparison with other machine learning models

In this section, we experiment on how the decision tree model performs compared with other popular machine learning models: logistic regression, support vector machine (SVM), Naive Bayes, k-nearest neighbors (KNN) and neural network. The 2017A dataset was used, with a random 70% of it as training data and the remaining 30% as testing data. The results are shown in Fig. 2.11. It can be seen that the decision tree model performs better than other models. The decision tree has 97.02% prediction accuracy and 1.24% false negative. Logistic regression and neural network perform close to decision tree. However, they are not as easily explainable as decision tree.

Considering that there are some recent work on rationalizing neural networks, we adapted and implemented a neural network model with reason generator [35] to have a comprehensive comparison with decision tree. The details of adaptation and implementation are omitted due to the space limitation and will be provided in an extended version of this paper. As shown in Fig. 2.11, the performance of neural network with reason generator is not as



good as decision tree. We then compare decision tree and neural network model in the reason code generated. The average length of reason codes generated by decision tree is about 4, while the average length of neural network is around 8.5. When reason codes are sufficient to support predictions, shorter reason codes are much better and easier to interpret/verify. Thus the reason codes of decision are easier to verify.

### 2.5.9 Remediation Analysis Delay and Cost

**Delay of remediation decision analysis** Here, we analyze the time saved by automating the decision analysis for  $Org_A$ . We first compute the time required by the machine learning-based framework. The vulnerability analysis and decision prediction time is in millisecond scale, which can be negligible. When the machine learning model is dynamically trained, it will require operators to verify 10% vulnerability predictions as discussed in Section 2.5.5. The average time of prediction verification is 28.8 seconds as shown in Section 2.5.4. For the 3476 vulnerabilities in the year covered by the 2017A dataset, the average verification time for  $Org_A$  is 2.3 personal hours per month (which is the typical decision cycle). Suppose there is only one operator. This can be completed within 2.3 hours, making the decision delay 2.3 hours.

Based on the test in Section 2.5.4, the average time of manually analyzing each vulnerability is 7 minutes. The total manual analysis time for  $Org_A$  is 33.8 person hours per month. Again, suppose there is only one operator. Theoretically, this task can be completed in 33.8 hours, making the decision delay 33.8 hours. However, in practice, when the time needed for a task is long, the total time span of completing the task will be more than simply the person hours. Human operators cannot work 24 hours a day like a machine and may take a rest now and then, need perform some other duties such as meeting and reporting, and can be distracted by other things like chatting with each other. Also, electric utilities like

*Org<sub>A</sub>* usually hold weekly meetings among security operators to perform remediation decision analysis. All of these factors can generate extra delay of remediation decision making. That might make the analysis process span across days and even weeks. This is validated by our survey described in Section 2.2, where 50% of participants indicated it takes them more than 16 days to complete remediation action analysis for each cycle.

According to this sketch analysis, the delay of completing remediation decision analysis with machine learning could be hours, but the delay with manual decision making could be days and even weeks. When remediation decisions for vulnerabilities can be made earlier, those high-risk vulnerabilities (that need to be patched now or mitigated now) can be identified earlier and hence remediated earlier, which will greatly reduce the security risks of electric utilities. Hence, utilities using our machine learning framework will face much less risks.

**Cost of remediation decision analysis** Since the VPM problem for security operations is one of human resource allocation, we analyze the personnel cost saving brought by machine learning. From the above delay analysis, it can be seen that with machine learning 31.5 person hours can be saved each month for *Org<sub>A</sub>*. That results in 378 person hours of saving per year. For larger utilities with more assets, the saving is even more. For example, in our survey, one participant company indicates it has 12,000 vulnerabilities per year. That would make the total saving to 1,305 person hours.

## 2.6 Related Work

There are many VPM solutions available for corporate use, such as GFI LanGuard [48], Patch Manager Plus by ManageEngine [9], and Patch Manager by SolarWinds [49]. However, these solutions focus on vulnerability discovery and deployment of patches rather than the decisions necessary to optimize resources for vulnerability remediation. Most so-

lutions designed for VPM in electric utilities also fall into this category, such as Doble Engineering’s PatchAssure [12], Flexera [13], or FoxGuard Solutions [50]. They are unable to analyze vulnerabilities against the operating environment and make prioritized decisions on how to address vulnerabilities.

When addressing vulnerabilities, there are some publicly-available sources of information. The NIST NVD [54] provides a well-structured, reliable data feed of vulnerabilities and their corresponding information as they are reported. Vulners [61] has a freely-accessible API to search vulnerability information and discover available exploits; the Exploit Database [11] also allows users to search for available exploits. Tenable [58] has recently released a tool which provides predictive analysis for exploitability of a security vulnerability. This tool focuses on the exploit features whereas we explore the relationship between the vulnerability and the asset to which it applies. The exploitability from the Tenable tool could be used as one feature of our machine learning framework.

There is also academic research on this topic. [46] and [14] analyze large vulnerability datasets and report trends in vulnerability characteristics, and disclosure and discovery dates. [36], [5], and [41] analyze patches and patch behavior, such as the windows of time between when patches are released and when they are installed or the effectiveness of a patch protecting against a vulnerability. [3], [60], and [62] describe approaches for prioritizing vulnerability patching based on the calculated severity of an exploit. [33] and [47] analyze the risks of network attacks based on attack graph. However, these works do not combine vulnerability metrics with organizations’ unique environment to analyze vulnerability remediation decisions. Machine learning has been applied to discover vulnerability in software and source code [64] [19] [16] [1] [45] [67], but our work uses vulnerability features with asset information to determine how to remediate vulnerabilities.

## 2.7 Discussions

Even though the proposed framework has only been tested on electric utilities, being consistent with the DHS guideline, it is general and can be applied to many other organizations especially critical infrastructures, which suffer from similar VPM challenges and constraints. The way of applying the framework to other organizations is similar to the application on electric utilities, but the identified asset features and remediation decisions might be different. In future work, we will test our framework on other types of organizations.

In some cases, operation contexts might affect remediation actions. For example, when one vulnerability outbreaks and makes to headlines (e.g., the Meltdown vulnerability), a company's administrators might want it to be remediated as soon as possible due to pressure from public reputation. Then operators might choose Patch-Now or Mitigate-Now-Patch-Later instead of Patch Later, even if the decision tree prediction based on vulnerability/asset features is Patch Later. In such cases, operators can use their decisions to override decision tree predictions. We will explore inclusion of such operation contexts in automation in future work.

## 2.8 Summary

This chapter addressed the need for more effective decision support to address VPM challenges and proposed a decision tree based framework to automate the analysis remediation decisions for vulnerabilities. We tested an instance of the framework customized for one electric utility over two datasets obtained from the utility. Results showed high prediction accuracy and time savings. These results demonstrate the value in applying machine learning to automate VPM processes. Even though the framework is only tested in electric utilities, it can easily adapt to other critical infrastructures.

## 3 Remediation Action Scheduling Optimization

### 3.1 Introduction

In Chapter 2, we have developed a machine learning method to predict how to address the newly discovered vulnerabilities. Every month, hundreds or even thousands of vulnerabilities to be remediated. However, not all vulnerabilities can be remediated quickly due to limited resources. It is important to carefully schedule the remediation order so that the total risk is minimal.

In this chapter, we will schedule the patching orders by considering vulnerabilities' impact and their probabilities of being exploited. One vulnerability's risk is affected by its impact on the system and its probability of being exploited. A vulnerability is becoming more and more likely to be exploited after it is discovered, which means the probability of being exploited is changing over time. Thus the vulnerability's risk is also changing over time. We will first predict probability  $p(t)$  that each vulnerability is exploited at time  $t$  with neural network method, and calculate each vulnerability's risk  $r(t)$  over time  $t$  based on its impact and  $p(t)$ . Then we will model this as an Mixed Integer Programs (MIP) problem to automatically derive optimal patch schedules.

### 3.2 Approach

The patching order can significantly affect the risk that the system is facing. As an example, a system has two vulnerabilities to remediate  $V_a$  and  $V_b$ . Assume  $V_a$ 's impact score is 9.5 with probability 0.6 of being exploited and  $V_b$ 's impact score is 6.0 with probability 0.8 of being exploited. It takes about 2 hours to patch  $V_a$  and 0.5 hour to patch  $V_b$ . If we prioritize the patches only by impact score,  $V_a$  will be first patched. Then the risk from  $V_a$  is:  $9.5*0.6*2$ , the risk from  $V_b$  is  $6.0*0.8*2.5$  and total risk is 23.4. If  $V_b$  is patched first, the

total risk is 16.65. Changing the patching order can significantly reduce the risk. In this section, we describe the approach to get optimal patching scheduling order by considering vulnerability impact, probability of being exploited and patching time. We will first predict the vulnerabilities' exploitability and then schedule the patching order.

### 3.2.1 Risk Metric

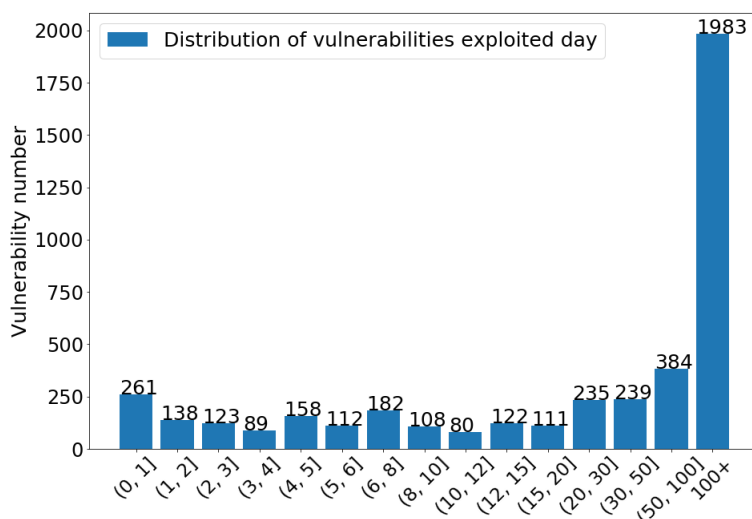
Here we will first describe how we measure the risk that a vulnerability poses to the system. Each vulnerability comes with Common Vulnerability Scoring System (CVSS) metrics, which shows the principal characteristics of the vulnerability and produce a numerical score reflecting its severity [10]. CVSS score can be used to quantify a vulnerability risk. However, it does not capture a vulnerability's probability of being exploited as time goes by, which only shows the risk at the time point when the vulnerability is published. Here we will define a new metric to quantify a vulnerability's risk by combining the vulnerability impact on the system and its probability of being exploited as stated in [56] [32]:

$$r(t) = I * p(t) \tag{3.1}$$

$r(t)$  is the risk of a vulnerability at time  $t$ ,  $I$  is the impact score on the system and  $p(t)$  is the probability to be exploited at time point  $t$ . The impact score can be calculated based on CVSS scoring system, where  $I = 10.41 * (1 - (1 - ConfImpact) * (1 - IntegImpact) * (1 - AvailImpact))$  as described in [27]. *ConfImpact*, *IntegImpact*, and *AvailImpact* are three characteristics defined in CVSS metrics to show the vulnerability's confidentiality, integrity and availability impact on the system respectively. Each characteristic has three different impact level, 'complete', 'partial' and 'none'. When its value is 'complete', it is numeralized as 0.660, 'partial' is numeralized as 0.275, and 'None' is numeralized as 0.0.

### 3.2.2 Exploitability Prediction

If a vulnerability is exploitable, it means that there are available exploit codes to leverage the vulnerability to launch attacks. The exploit can be released any time after or even before a vulnerability is newly disclosed. Knowing how soon the vulnerabilities is likely to be exploited can help in scheduling when to patch the vulnerability. In this section, we will predict the probability of a vulnerability to be exploited in the  $n_{th}$  day since it is published with feed forward neural network (FFNN).



**Figure 3.1:** Distribution of vulnerabilities’ exploited day after being published

We collected 67965 vulnerabilities from Vulners database [61], 46380 out of which have no exploits, 17260 already have exploits before vulnerabilities are published, and 4325 vulnerabilities are exploited after they are published. For these vulnerabilities whose exploits are released before they are published, it is not necessary to do prediction since their exploitability are already known. We only need predict the 4325 vulnerabilities whose exploits are unavailable when they are published. The vulnerability exploitability distribution is shown in Fig. 3.1. The X axis shows in which day the vulnerability is exploited after being published and the Y axis shows the number of vulnerabilities exploited in that day.

For example, about 250 vulnerabilities are exploited in day 7th or 8th after being published and around 2000 vulnerabilities are exploited after 100 days. To keep the dataset’s balance, we also randomly sampled 4325 vulnerabilities from the unexploitable vulnerabilities as the experiment dataset, which will be used as training data to train the neural network model.

**Table 3.1:** Extracted Features

<b>Feature Family</b>	CVSS metric	CWE	software name	description	title	modify date - publish date	last seen date - publish date
<b>Number</b>	8	1	1	3000	500	1	1

**Feature Extraction:** To train the neural network model, we need select import features to represent each vulnerability. CVSS metric is used as features which are attack vector, attack complexity, user privilege, user interaction, integrity, availability, confidentiality and CVSS score. Common weakness enumeration (CWE) which shows the vulnerability type, and software name are also used as features. Each vulnerability has its description and title which also provide valuable information. Since descriptions and titles are descriptive texts, we will apply tf-idf (term frequency- inverse document frequency) [59] to extract important words from description and title texts as features. Tf-idf is able to identify important words and deprive stop words (e.g. the) and common words appearing in most descriptions or titles such as “CVE”. We extract bi-grams as features, which has better performances than uni-gram in this prediction task. 8500 bi-grams are extracted from descriptions and 1470 bi-grams from titles. Then we apply random forest to select top 3000 most important features from descriptions such as “windows server”, “execute arbitrary”, “verify certificates”, “spoof servers”, “access restriction” and top 500 most important features from titles. Besides, each vulnerability has publish date, modify date and last seen date. The time differences between modify and publish date, last seen date and publish date are highly related to exploitability. Thus we also take the time difference between modify date and publish date, and difference



between last seen date and publish date used as features. The feature number is shown in Table 3.1.

**Exploitability Prediction with Neural Network:** The historical vulnerability data can be used as the training data to train the FFNN model. When a new vulnerability is published, it can be fed into the trained FFNN model which will predict when the vulnerability will be exploited and provide the probability of being exploited on each day. When we predict the exploitability probability of  $n_{th}$  after being published, we will relabel the training dataset so that the vulnerabilities which are exploited under  $n$  day are labeled as 1 and the ones which are exploited over  $n$  day or unexploitable are labeled as 0. Then this relabeled dataset is used to train a model and this model can provide the probability of being exploited in  $n_{th}$  day.

### 3.2.3 Patch Schedule Optimization

Different vulnerabilities have different probabilities of being exploited and risk level on each day. It is necessary to schedule which vulnerabilities should be patched first so that the total risk that the system is facing is as low as possible. In this section, we will model the problem as an Mixed Integer Programming problem (MIP), which is a mathematical optimization problem in which some or all of the variables are integers, and discuss how to get the optimal patching order with optimization algorithm.

Let  $v_i$  denote vulnerability  $i$ ,  $p_i(t)$  denote the probability of being exploited for vulnerability  $v_i$  over time  $t$ ,  $I_i$  denote impact score of  $v_i$ , and  $d_i$  denote execution delay to process patch  $i$ . Our objective is to schedule the patch order so that the total risk that a system is facing is as low as possible. The risk of an unpatched vulnerability at time  $t$  is  $r(t)$  as defined in Equation 3.1. Since a vulnerability cannot be exploited after it is patched, this vulnerability will not pose any risk to the system. Thus the total risk caused by a vulnerabil-

ity is the integral of  $r_i(t)$  between its publishing time and the patching time:  $\int_0^{s_i+d_i} r_i(t)dt$ , where  $s_i$  is the time point when patch starts. In order to model this as an MIP problem and apply optimization algorithm, we convert this formula to discrete calculation which is:  $\sum_{t=0}^{t=s_i+d_i} r_i(t)$ . In this work, we consider 30 minutes as a unit,  $d_i = 2$  means it takes 2 time units to complete patch, which is 60 minutes.  $s_i = 10$  means starting patch at time unit 10, which is 5 hours. For all the vulnerabilities, the total risk is:  $\sum_{i=0}^n \sum_{t=0}^{s_i+d_i} r_i(t)$ . The goal is to minimize the total risk that unpatched vulnerabilities pose to the system.

In practice, assets are divided into different asset groups based on the assets' functions and physical locations for easier management as we discussed in Chapter 2.3.2. Operators need coordinate to schedule downtime for machines before applying patches. Thus, once an asset group's machines are scheduled down, all the patches applicable to the asset group will be applied in succession. Thus this optimization problem can be implemented into two phases. In the first phase, we will take one asset group's vulnerabilities as a task and determine which group should be patched first. In the second phase, schedule the vulnerability patching order for each group.

**Phase I: Asset group patching order optimization:** In this phase, each asset group is taken as one task. We use  $R_j(t)$  to denote the risk of all vulnerabilities in asset group  $j$  over time  $t$  and  $D_j$  is the execution delay to process all vulnerabilities in asset group  $j$ , where  $R_j(t) = \sum_{i=0}^{i=n_j} r_i(t)$  and  $D_j = \sum_{i=0}^{i=n_j} d_i$ .  $n_j$  is the vulnerability number in asset group  $j$ . In an organization, there are usually several operators to apply patches. And one operator can be responsible for one or several asset groups, but he can only work on one patch at a time. We use  $X_{(j,a)}$  to denote that asset group  $j$  is assigned to operator  $a$ . The asset group scheduling optimization problem can be formulated as:

---

---

**Inputs:**

$J$  set of all asset groups  $j \in J$  in the organization

$A$  set of all available operators  $a \in A$  to apply patches

$D_j$  execution delay to process all patches in asset group  $j$

$R_j(t)$  risk of all vulnerabilities in asset group  $j$

$S_j$  scheduled start time of asset group  $j$

$X_{(j,a)}$  allocation of asset group  $j$ ,  $X_{(j,a)} = 1$  if operator  $a$  is assigned to asset group  $j$ ; otherwise 0,  $X_{(j,a)} = 0$

$\theta_{(j,k)}$  support variable used to determine whether the processing time of asset group  $j$  and  $k$  are overlapped for each operator.  $\theta_{(j,k)} = 1$  if asset group  $k$  is started before  $j$  is completed, otherwise  $\theta_{(j,k)} = 0$

**Goal:**

**Minimize:**  $\sum_{j \in J} \sum_{a \in A} X_{(j,a)} \sum_{t=0}^{S_j + D_j} R_j(t)$

**Subject to:**

- $\sum_{a \in A} X_{(j,a)} = 1$  (each patch must be assigned once to exactly one operator)
  - $S_k \geq S_j + D_j$  (a patch cannot start until its predecessor are completed)
  - $X_{(j,a)} + X_{(k,a)} + \theta_{(j,k)} + \theta_{(k,j)} \leq 3$  (if asset group  $j$  and  $k$  are assigned to the same operator, their execution times cannot overlap)
- 

**Phase II: Patching order optimization in each asset group:** After getting the asset group patching order in first phase, we will schedule the patching order for each group. We can get each asset group's patch start time  $S_j$  from first phase's scheduling. Then all the applicable vulnerabilities in this asset group should be patched after  $S_j$ . The formulated optimization algorithm is described as following:

---

---

for each asset group do:

**Inputs:**

$I$  set of applicable vulnerabilities  $i \in I$  in the asset group

$d_i$  execution delay to process vulnerability  $i$

$r_i(t)$  risk of vulnerability  $i$

$s_i$  scheduled start time of vulnerability  $i$

$\theta_{(i,k)}$  support variable used to determine whether the processing time of vulnerability  $i$  and  $k$  are overlapped.  $\theta_{(i,k)} = 1$  if asset group  $k$  is started before  $i$  is completed, otherwise  $\theta_{(i,k)} = 0$

**Goal:**

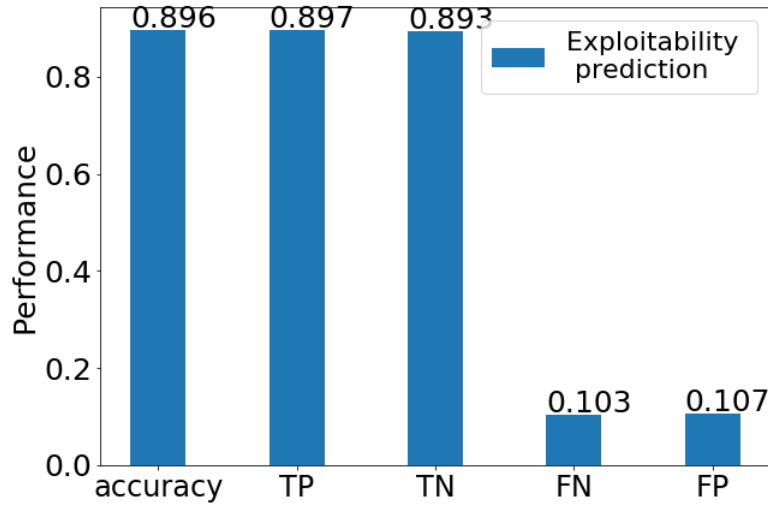
**Minimize:**  $\sum_{i \in I} \sum_{t=0}^{s_i+d_i} r_i(t)$

**Subject to:**

- $\theta_{(i,k)} = 1$  (a patch cannot start until other is completed)
  - $s_i \geq S_j$  (vulnerabilities can be patched after its asset group start time  $S_j$ )
- 

### 3.3 Evaluation

#### 3.3.1 Evaluation on Exploitability Prediction



**Figure 3.2:** Exploitability Prediction

The dataset used in this experiment consists of 8650 vulnerabilities, half of which are exploited and half are unexploited. We split the dataset into two part: 67% as training data and the remaining as testing data. We implemented the FFNN model with Python sklearn. We first test the performance of exploitability prediction without considering which day the vulnerabilities are exploited. No matter whether the vulnerability is exploited in the first day or in 500th day after being published, it will be labeled as exploited. The prediction accuracy is shown in Fig. 3.2. It has 89.6% prediction accuracy and the true positive (TP) is 89.7%. The true negative (TN), false negative (FN) and false positive (FP) are 89.3%, 10.3%, and 10.7% respectively.

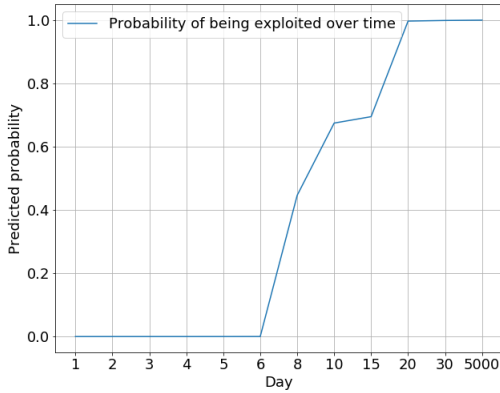
Then we test the performance of predictions on which day the vulnerability will be predicted. For each day's exploitability prediction, the dataset is relabeled as discussed in Section 3.2.2 and then used to train a specific neural network model for this day. The prediction accuracy is shown in Table 3.2. As the time goes by, the prediction accuracy is becoming better. For example, the prediction for 1st day is 74.7%, and the prediction for 30th day is 82.0%. This is because the longer the time after being published is, the more vulnerabilities are exploited. Thus the training dataset is becoming larger, the model will be better trained and its performance will also become better.

The FFNN model also predicts the probability of being exploited  $p(t)$  in each day. We give two examples of two vulnerabilities ( $v_i$  and  $v_k$ ) predicted probabilities of being exploited which are shown in Fig. 3.3 and Fig. 3.4. The X axis shows the  $n_{th}$  day after being published and Y axis means the probability of being exploited at that very day. In reality,  $v_i$  was exploited in the eighth day after being published and  $v_k$  was exploited in the sixth day after being published. From the predicted result shown in the figures, it can be seen that  $v_i$ 's probability of being exploited is around 0.5 in 8th day and  $v_k$ 's predicted probability in 6th day is around 0.6. The predicted probabilities can imply when the vulnerabilities will

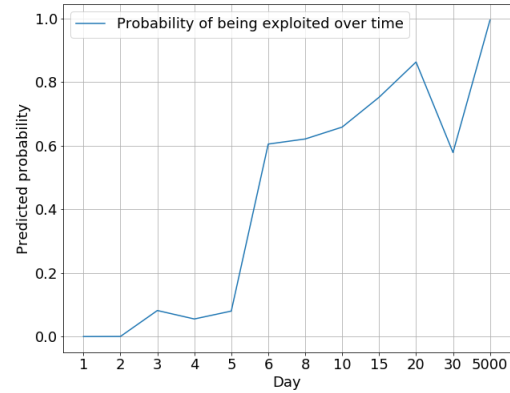
**Table 3.2:** Exploiting time prediction

Day	Accuracy	TP	FN	TN	FP
1	0.747	0.742	0.258	0.753	0.247
2	0.783	0.772	0.228	0.795	0.205
3	0.784	0.768	0.232	0.803	0.197
4	0.799	0.788	0.212	0.810	0.190
5	0.801	0.789	0.211	0.814	0.186
6	0.802	0.789	0.211	0.818	0.182
8	0.804	0.784	0.216	0.827	0.173
10	0.809	0.809	0.191	0.808	0.192
15	0.816	0.801	0.199	0.834	0.166
20	0.812	0.81	0.19	0.818	0.182
30	0.820	0.819	0.181	0.821	0.179
5000	0.896	0.898	0.102	0.893	0.107

be exploited.



**Figure 3.3:** Predicted probability of vulnerability  $v_i$



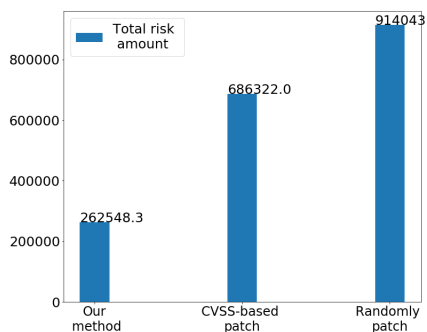
**Figure 3.4:** Predicted probability of vulnerability  $v_k$

### 3.3.2 Evaluation on Patch Scheduling

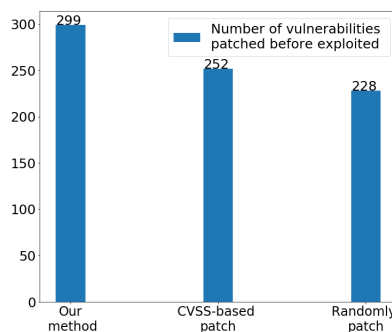
In this section, we will test how our patch scheduling optimization method performs. The scheduling optimization problem is implemented with around 2000 lines of Python codes by using OR-Tools library, which is developed by Google for optimization problems [43]. In order to know how our optimization method performs, we compare it with other

two commonly used patching methods in practice, patching randomly and patching based on CVSS score (patch vulnerabilities with higher CVSS score first). Both methods are implemented in Python. For the patching randomly method, we randomly select which asset group should be patched first and then for each asset group, randomly determine the vulnerabilities' patching order. For the CVSS score-based patching method, add all the applicable vulnerabilities' CVSS score up for each asset group and patch the asset group with higher CVSS score first. Then for each asset group, determine patch order based on CVSS score.

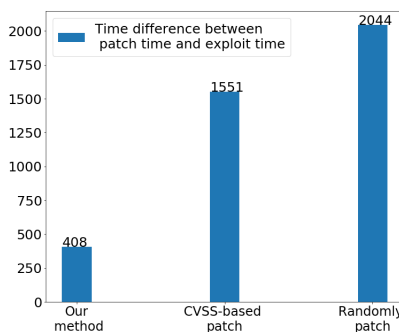
We compare these three method through three metrics. The first metric is the total amount of risks that unpatched vulnerabilities pose to the system, which can be calculated as  $\sum_{i=0}^n \sum_{t=0}^{s_i+d_i} r_i(t)$ , and  $n$  is the number of all vulnerabilities. The second metric is the number of the vulnerabilities that are patched before being exploited. For those vulnerabilities which are not patched before being exploited, we will calculate the time difference between the patching time and exploited time to see how long the system is exposed to the exploited vulnerabilities, which can be calculated as  $\sum_i t_{patch\_time} - t_{exploit\_time}$  if  $t_{patch\_time} > t_{exploit\_time}$ , where  $t_{patch\_time}$  is the patching time and  $t_{exploit\_time}$  is the exploited time.



**Figure 3.5:** Total amount of risks of unpatched vulnerabilities under one operator



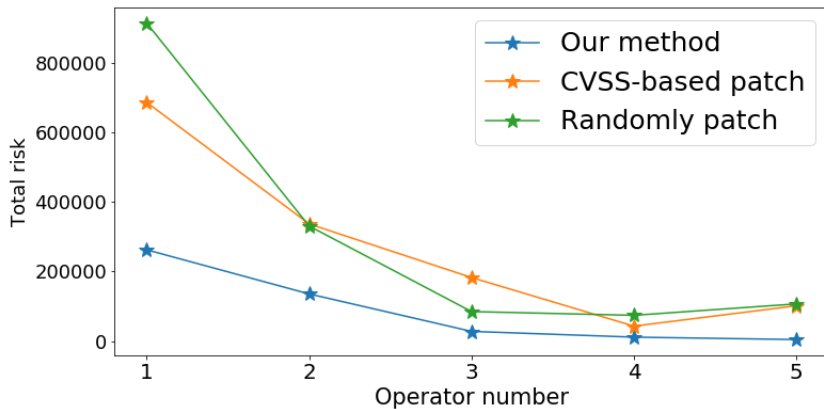
**Figure 3.6:** Number of vulnerabilities patched before exploited under one operator



**Figure 3.7:** Time difference between patch time and exploit time if patched after exploited under one operator

We use the real data, the installed software and asset group information, from the utility company *Org<sub>A</sub>* as the testing data. We extracted 390 applicable vulnerabilities published in January 2019 through the software names. Then we predict the exploiting probability for these retrieved vulnerabilities with our trained neural network model. Then we run our scheduling optimization algorithm to get optimal patching order. There are totally 26 asset groups in the testing data.

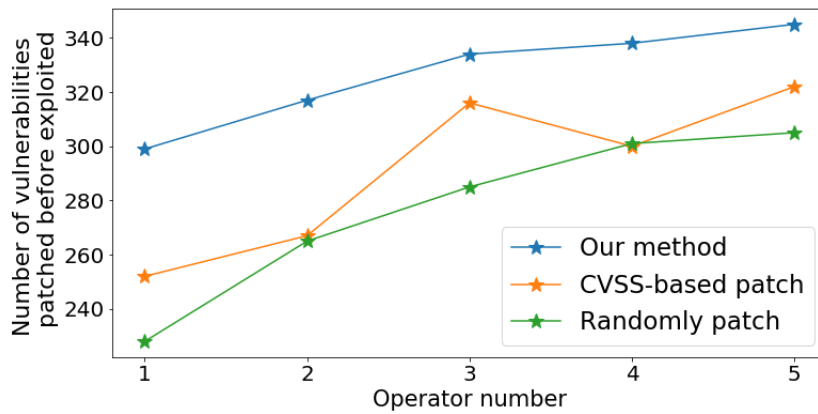
We first test the performances when there is only one operator in the organization. The results are shown in Fig. 3.5, 3.6 and 3.7. It can be seen that our optimization method performs much better than the other two method. The total amount risks can be reduced from 686322 to 262548 when compared with CVSS-based method and from 914043 to 262548 when compared with randomly patch method. The risk is decreased by at least 60%. Our scheduling method can patch 299 vulnerabilities before being exploited, while CVSS-based and randomly patching methods can patch 252 and 228 vulnerabilities. For these vulnerabilities that are not patched before exploited, they should be patch as soon as possible. Fig. 3.7 shows that our method expose the system to the exploited vulnerabilities for the shortest time, which is 1636 hours less than CVSS-based method and 1143 hours less than randomly patching method.



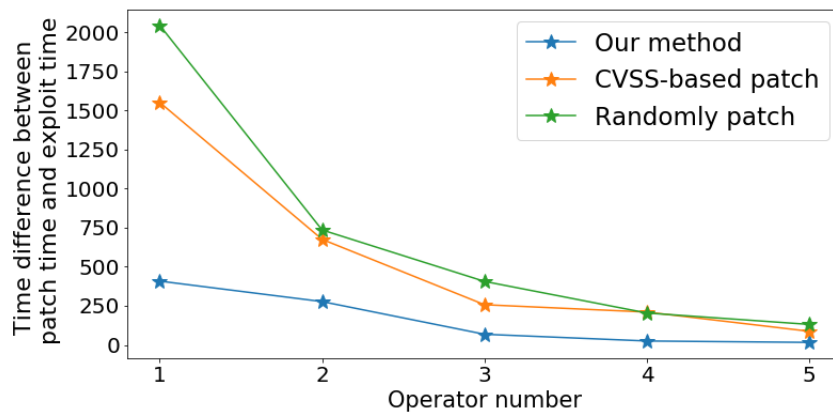
**Figure 3.8:** Total amount of risks of unpatched vulnerabilities under multiple operators



Then we test the performances when there are multiple operators in one organization. The results are shown in Fig. 3.8, 3.9 and 3.10. With more operators, the total risk becomes less because more patches can be processed earlier and timely. The results show that our optimization method performs much better than the other two methods under multiple operators.



**Figure 3.9:** Number of vulnerabilities patched before exploited under multiple operators



**Figure 3.10:** Time difference between patch time and exploit time if patched after exploited under multi operators

### 3.4 Related Work

To the best of our knowledge, there is no work to schedule the patching order by considering the vulnerabilities' dynamic probabilities of being exploited so that the total risk that a power system is facing is minimal. There have been some work about patching prioritization. The work in [15] prioritized the vulnerability remediation by quantifying the vulnerabilities' risk based on CVSS temporal and environmental scores. The work in [23] prioritized the vulnerabilities with two security metrics, system risk and attack cost. The work in [17] [39] [4] determine the patching order through game theory in an attack/defense scenario. These works do not consider the vulnerabilities' dynamic nature and the probability of being exploited is changing over time. They assume that a vulnerability's risk to a system is constant, and thus the patching order does not guarantee the system's total risk is minimal.

### 3.5 Summary

In this chapter, we propose a method about how to schedule the patching order to reduce the risk that vulnerabilities pose to the system. We first predict the vulnerabilities' probability of being exploited over time. Each vulnerability's risk can be calculated based on its impact score on the system and the predicted probability. Then we model the scheduling problem as MIP and get the optimized scheduling patching order so that the total risk is minimum. The evaluation based on a utility company's real data shows our method can significantly reduce the risk.

This method is the following step of the work in Chapter 2. After we determine how to address the vulnerabilities (patch, mitigating, patch later), we can first apply the optimization method on the group of vulnerabilities that need to be patched and mitigated, to determine the patching and mitigation order. Then we can apply this method on the

vulnerabilities that can be patched later and get their optimized scheduling order.

## 4 Identifying Exploited Vulnerabilities through Data Forgery Attack Detection and Localization

### 4.1 Introduction

It is important for electric utilities to know whether some vulnerabilities have already been exploited specifically for their power system to help them better schedule remediation actions. For example, if a vulnerability is exploited, it has to be remediated immediately. Different vulnerabilities may require different identification methods. Here, we explore identifying exploited vulnerabilities by detecting and localizing false data injection attacks and give a case study in the Automatic Generation Control (AGC) system.

AGC is a key control system in the power grid which aims to keep balance between power generation and load and maintain a stable power system frequency. It automatically adjusts power generation in response to area control imbalance. In AGC, a control center periodically collects the power system's frequency and tie-line power flow measurements from meters to calculate Area Control Error (ACE) and change the set-point of the generator governors participating in AGC based on the calculated ACE. If malicious tie-line power flow or frequency data is injected to normal measurement to mislead AGC to do miscalculations, AGC may issue wrong commands based on the miscalculated ACEs.

False data can be injected to the system in two ways. One way is to directly inject the false data packets to the communication network channel. However, such attacks can be easily detected by authentication techniques. The other way is to compromise and control the devices such as meters to launch such attacks. If there are some unaddressed vulnerabilities existing in a device, attackers can exploit the vulnerabilities to compromise the device. The increasing number of smart devices in the power grid and the connection to external networks make this way more feasible. If the false injected data are carefully designed,

existing methods implemented in control centers such as State Estimation (SE) and Bad Data Detection (BDD) [18] are unable to detect intelligent cyber attacks. The work in [63] has shown that the existing schemes can be bypassed by carefully designed false data injection attacks.

If the attacks can be detected and localized, we will know which devices and vulnerabilities are exploited, and thus can better prioritize the vulnerability remediation. In this work, we propose Neural Network-based and Fourier Transform-based approaches to detect and localize false data injection attacks in AGC. In the first approach, we adopt Long Short Term Memory (LSTM) neural network to learn ACE patterns from historical data, predict ACE sequence pattern for next detection window based on the learned patterns, compare the predictions with the corresponding ACE sequence pattern calculated from measurements to determine whether there is forged data in the sequence. The LSTM model can be built with single feature: the historical ACE data (single-feature LSTM). It can also include more related features (i.e. multi-feature LSTM), such as frequency and tie-line power flow, to achieve better performance. The LSTM-based approach is also developed to localize attacks by checking which measurement is abnormal so that we can know which meter is compromised. In the second approach, we convert ACE and measurement data from the time domain to the frequency domain by using Fourier Transform and then check if ACE data is normal in the frequency domain. We test these approaches on both simulated and real datasets. Since we only use historical data that are already available in current AGC systems, our detection methods can be easily deployed without interrupting normal services.

This chapter is organized as follows. Section 4.2 describes the AGC system model and five attack models considered in this paper. Section 4.3 describes the proposed detection and localization methods. Section 4.4 presents the simulated system and simulated and real dataset. Section 4.5 shows the performance of the proposed methods on both simulated

dataset and real dataset. Section 4.6 discusses related work about false data injection attack detection in AGC. The last section concludes this paper.

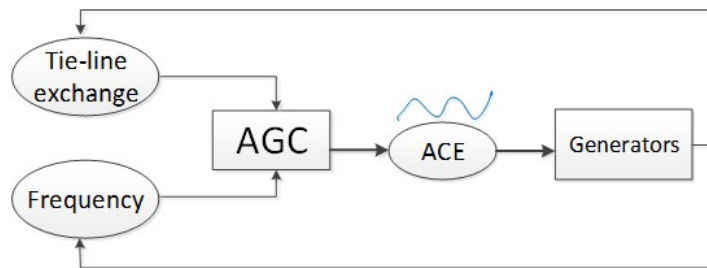
## 4.2 Preliminary

### 4.2.1 AGC System

For a control area, AGC is used to adjust power generation to maintain the frequency at the scheduled value. AGC is an automated control system and its workflow is illustrated in Fig. 4.1. It periodically (every 2-4 seconds) receives the measurements of frequency from this control area and tie-line power flows between this area and neighboring areas from field devices, and calculates the ACE according to the equation

$$ACE = (P_{tie} - P_{sch}) + B(f - f_{sch}) \quad (4.1)$$

where  $P_{tie}$  and  $f_{sch}$  are the scheduled tie-line power flow and the scheduled frequency respectively.  $B$  is the frequency bias factor, which is constant for each power system and is estimated annually. Then AGC adjusts the power generation of generators according to the obtained ACE.



**Figure 4.1:** AGC System

### 4.2.2 Attack Models

To cause miscalculations of ACEs, attackers can inject false data to frequency measurements or tie-line power flow measurements. Normally, ACE is within a specific range. Once ACE exceeds the limit, an alarm will be triggered to human operators and the AGC may be suspended [30]. If the attacker injects a significant attack to the measurements, it might be detected by SE or BDD, or trigger the alarm. Thus the attacker might only modify the measurements slightly so that the attacks are not detected. However, one stealthily forged data measurement may not be enough to introduce significant impact to the power grid. According to [57], the shortest time to stealthily mislead the system frequency to breach the safety condition without triggering AGC suspension is at least 10 AGC cycles, which means that in order to achieve expected effects, the attacker needs to inject a series of false data to indirectly control the generator for a period of time.

One way to launch stealthy attacks is to find the maximum and minimum values of normal frequency or tie-line power flow measurements and then inject the minimum or maximum data into the measurements. Since the maximum and minimum values are still within normal range, they will be trusted by AGC. In this work, these two types of attacks will be explored. Let  $T_a$  represent the attack period,  $t$  represent time,  $t_0$  represent the time point when the attacker starts an attack,  $y(t)$  represent the true measurement (which could be either frequency or tie-line power flow as discussed later) value without attacks, and  $y^*(t)$  represent the measurement value with possible attacks.

**Max Attack:** This attack replaces the measurements with the maximum data value  $y^{max}$ .

$$y^*(t) = \begin{cases} y(t), & \text{if } t \notin T_a \\ y^{max}, & \text{if } t \in T_a \end{cases} \quad (4.2)$$

**Min Attack:** This attack replaces the measurements with the minimum data value

$y^{min}$ .

$$y^*(t) = \begin{cases} y(t), & \text{if } t \notin T_a \\ y^{min}, & \text{if } t \in T_a \end{cases} \quad (4.3)$$

In addition to the Max and Min attacks, we also consider three attack models which are also explored in [51, 24]: scaling attack, ramp attack, and random attack. In these attack models, the attacker keeps launching attacks until achieving expected results. The three attack models can be described as follows.

**Scale Attack:** The attack scales measurement values up or down by multiplying with a scaling parameter  $\lambda_s$ .

$$y^*(t) = \begin{cases} y(t), & \text{if } t \notin T_a \\ y(t) + \lambda_s \cdot y(t), & \text{if } t \in T_a \end{cases} \quad (4.4)$$

**Ramp Attack:** The attack gradually modifies measurements by adding  $\lambda_r(t - t_0)$ .  $\lambda_r$  is a ramping parameter. This type of attack is more difficult to detect because it has very small and unnoticeable changes at the beginning of the attack period.

$$y^*(t) = \begin{cases} y(t), & \text{if } t \notin T_a \\ y(t) + \lambda_r(t - t_0), & \text{if } t \in T_a \end{cases} \quad (4.5)$$

**Random Attack:** The attack modifies measurement values by adding some random values in a range with lower bound  $\lambda_a$  and upper bound  $\lambda_b$  during the attack period.

$$y^*(t) = \begin{cases} y(t), & \text{if } t \notin T_a \\ y(t) + \text{rand}(\lambda_a, \lambda_b), & \text{if } t \in T_a \end{cases} \quad (4.6)$$

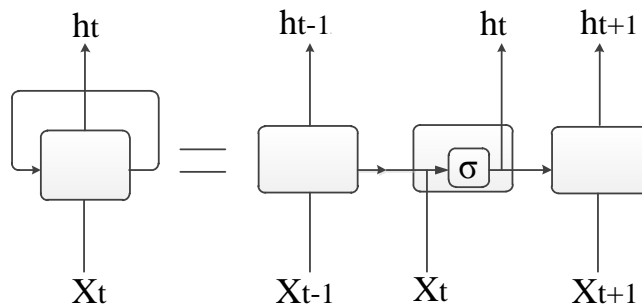
For the same power system, the higher the attack parameters  $\lambda$  are, the more significant the attacks are. However, attacks with same parameters may have different impacts on different power systems. For example, for two different power system  $A$  and  $B$ , the average



ACE of  $A$  is 800MW while  $B$ 's average ACE is 40MW. If a random attack within range  $(0, 40)$  attacks system  $A$  and  $B$ , it only change  $A$ 's ACE by 5% while change  $B$ 's ACE by 100%.

### 4.3 Detection Methods

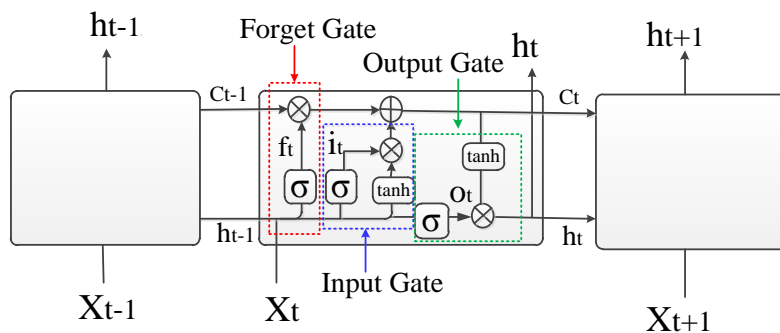
#### 4.3.1 LSTM Neural Network



**Figure 4.2:** Structure of Recurrent Neural Network

Based on the dataset observations, the ACE data of a control area have some patterns determined by the physical configuration of the AGC system (e.g., how ACE responds to load changes). Fig. 4.4 shows the ACE data pattern of 250 cycles from the real dataset PJM [44]. From the figure, it can be seen that the ACE data has similar patterns. For example, the data sequence pattern from cycle 0 to 49 is similar to the sequence pattern from cycle 50 to 100. If an attacker injects artificial data into AGC, the resulted ACE patterns will be different. Therefore, we can detect attacks through checking whether there are ACE data patterns that deviate from the normal patterns. Following this idea, we use neural network to learn the normal pattern of ACE time series and use it to detect attacks. To determine whether the pattern of the current data sequence is normal or has appeared before, the neural network model need have the ability to link the current observations with

the past observations. Recurrent Neural Network (RNN) is designed to deal with data with dependency [40], whose structure is show in Fig. 4.2. RNN can be regarded as multiple copies of the same neural network which are connected successively. The output of a neural network will be passed to its successor.  $X_{t-1}$  is the data at time point  $t - 1$  and its output data  $h_{t-1}$  will be passed to next neural network and taken as the input together with  $X_t$ . The output is calculated as:  $h_t = \sigma(W_h X_t + U_h h_{t-1} + b)$ , where  $W_h, U_h, b$  are the parameters and  $\sigma$  is the activation function in the neural network. In such way, the RNN model allows the information to be passed and persist.



**Figure 4.3:** Structure of Long Short Term Memory Network

However, RNN is not good at addressing long-term dependency. As the time sequence moves forward, the previous information carried by the neural network will become less and less, and eventually vanish. However, in our problem, we need to find what patterns the current observations follow, which requires the neural network remember and relate to previous patterns even far from current time point. In order to solve the long-term dependency problem, Long Short Term Memory (LSTM) [21] is adopted, whose structure is shown in Fig. 4.3.

Different from RNN, in LSTM, 'forget gate' and 'input gate' are added in each neural network chunk. The forget gate is to decide what information need to discard. The input

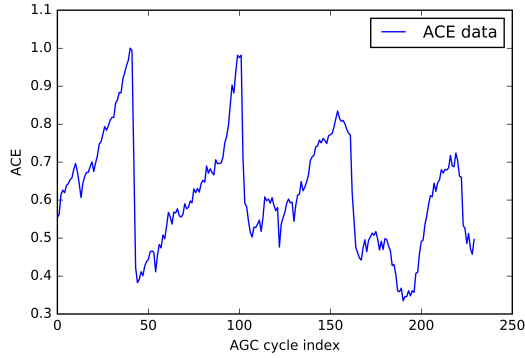
gate is to decide what new information are going to be stored. Thus these gates helps remove useless information while remember useful information for a long period of time. As shown in Fig. 4.3, not only  $h_{t-1}$  is passed to next neural network, but also  $C_{t-1}$ .  $C_t$  is a memory cell to store remembered information. The output  $h_t$  and  $C_t$  are calculated as follows:

$$\begin{aligned}
 f_t &= \sigma(W_f X_t + U_f h_{t-1} + b_f) \\
 i_t &= \sigma(W_i X_t + U_i h_{t-1} + b_i) \\
 C' &= \tanh(W_c X_t + U_c h_{t-1} + b_c) \\
 O_t &= \sigma(W_o X_t + U_o h_{t-1} + b_o) \\
 C_t &= C_{t-1} * f_t + i_t * C' \\
 h_t &= O_t * \tanh(C_t)
 \end{aligned} \tag{4.7}$$

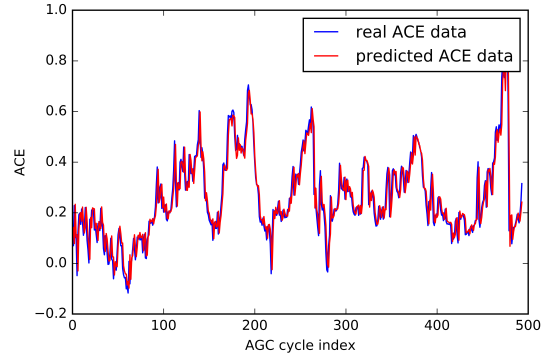
Where  $W, U, b$  are the neural network weight parameters. The important information will flow through the memory cell  $C_t$ . For our problem, LSTM is able to remember and find what previous sequence pattern that the current sequence resembles or is similar to. Then it can predict the next data sequence pattern based on the resembled pattern as well as the future measurements.

### 4.3.2 LSTM-based Detection Method

Our experiments on the PJM dataset show that the LSTM model trained by historical ACE time series data is able to make very accurate predictions for the next ACE sequence. We split the PJM ACE data into two parts, training data and testing data. Then training dataset (size: 1 million records) is used to train the model and then the testing dataset is fed into the model to do the prediction for each data point. The prediction results are shown in Fig. 4.5. The red ones are predicted data which fits the real data (blue ones) very well. Since the LSTM model has such high prediction accuracy, we can compare the predicted



**Figure 4.4:** ACE data pattern



**Figure 4.5:** Prediction with LSTM

ACE sequence with the measured ACE sequence to detect abnormal measurements. In each comparison, we calculate the distance between a predicted ACE data sequence with the corresponding measured ACE sequence by using Manhattan Similarity.

*Single-Feature LSTM:* we can only use past ACE data as features, which is  $X_t = \{ACE_t\}$ , and learn ACE patterns with LSTM model from historical ACE data, as we proposed in our previous work [65]. The experiments showed that this method has promising results on random and ramp attacks, but does not perform well on scale attack detection since scale attacks do not change ACE patterns, just scaling ACE values up and down.

*Multi-Feature LSTM:* in the single-feature LSTM model, only historical ACE are utilized to make predictions. Actually, in addition to past ACEs, the future ACE value also depends on frequency and tie-line power flow measurement value according to Equation 4.1. Furthermore, the frequency and tie-line power flow are affected by the variance between real load and power generation as we introduced in 4.1. And power generation amount is determined by forecast load. Thus the ACE values are closely related to frequency, tie-line power flow, real load and forecast load values. In order to predict ACE more accurately, we will consider these four factors in addition to past ACE data. The input of each time point is  $X_t = \{f_t, P_t, R_t, L_t, ACE_t\}$ , where  $f_t, P_t, R_t, L_t, ACE_t$  are the frequency, tie-line power flow, real load, forecast load and ACE at time point  $t$  respectively.

In particular, the LSTM model is first trained with historical data and can be updated dynamically (e.g., every day or every week) to include the newly generated data. We can feed the input data sequence into the trained model to do the prediction. The length of the input data sequence can be adjusted based on the datasets to achieve better prediction results. Suppose the input length is  $m$ , and  $X_t$  is the  $t$  time point data.  $X_t$  can be single feature  $\{ACE_t\}$  or multiple features  $\{f_t, P_t, R_t, L_t, ACE_t\}$ . We use the input data sequence  $(X_{(t+1)}, X_{(t+2)}, \dots, X_{(t+m)})$  to predict  $X_{(t+m+1)}$ . When predicted data sequence with  $n$  data points is available, it is compared with the measured data sequence at the same time points to check whether the measured data sequence deviates significantly from it. The detailed steps of the method are shown as follows which are run every  $n$  ACE cycles.

Step 1: Predict the next data sequence with the trained model.

$$\begin{aligned}
 X_{t-m+1}, X_{t-m+2}, \dots, X_t &\rightarrow \hat{X}_{t+1} \\
 X_{t-m+2}, X_{t-m+3}, \dots, X_{t+1} &\rightarrow \hat{X}_{t+2} \\
 &\dots \\
 X_{t-m+n}, X_{t-m+n+1}, \dots, X_{t+n-1} &\rightarrow \hat{X}_{t+n}
 \end{aligned}$$

Step 2: Calculate the distance between measured data sequence with predicted one with Manhattan Similarity.

$$d = \frac{1}{n} \sum_{i=1}^n |ACE_{t+i} - \hat{ACE}_{t+i}|$$

Step 3: Compare the distance with threshold  $\theta$ . It is normal data if the distance is less than the threshold  $d < \theta$ . Otherwise, it is regarded as attacked data.

### 4.3.3 LSTM-based Attack localization

After detecting existence of attacks, it is also desired to know where the attacks come from or which sensor is compromised. In the detection method, we detect attacks by finding

abnormal ACE patterns. Following this idea, we can also detect which measurement is attacked by checking the measurements' patterns. To localize attacks, we not only predict ACE values, but also predict the frequency and tie-line power flow value through the LSTM model. When abnormal ACE data is detected, we will compare the predicted frequency and tie-line power flow values with the real frequency and tie-line power flow measurements.

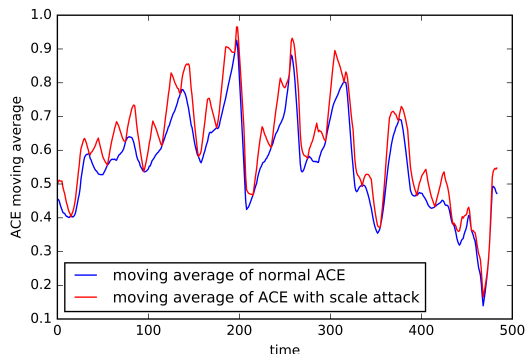
$$d_f = \frac{1}{n} \sum_{i=1}^n |f_{t+i} - \hat{f}_{t+i}|$$

$$d_P = \frac{1}{n} \sum_{i=1}^n |P_{t+i} - \hat{P}_{t+i}|$$
(4.8)

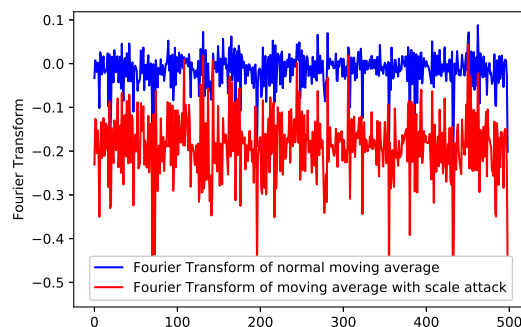
If  $d_f$  or  $d_P$  is larger than the thresholds, it means this measurement is under attack.

#### 4.3.4 Fourier Transform-based Method Detection and Localization

The single-feature LSTM-based method cannot detect scale attacks effectively since the scale attack just scales the data's value up or down and does not change the data sequence patterns. We propose another method, Fourier Transform-based detection, to complement single-feature LSTM-based detection.



**Figure 4.6:** Moving Average of Normal ACE and ACE with Scale Attack



**Figure 4.7:** Fourier Transform of Moving Average with scale attacks

Fourier Transform [7] can convert data from time domain to frequency domain to

observe data patterns. The moving average is the average of its previous  $m$  data to smooth fluctuations and highlight patterns which can make data sequence patterns more obvious. Thus we can calculate the moving average of data and then convert its moving average to frequency domain to observe patterns. If some unexpected changes occur, these can also be reflected on moving average. The moving average of attacked ACEs and normal ones are shown in Fig. 4.6. The blue line shows normal data's moving average and the red one is the moving average with the scale attack ( $\lambda_s = 0.2$ ) when the length of one attack period is 10 AGC cycles. It can be seen that the patterns of moving average is more obvious and attacked data's moving average is more fluctuated than the normal ones.

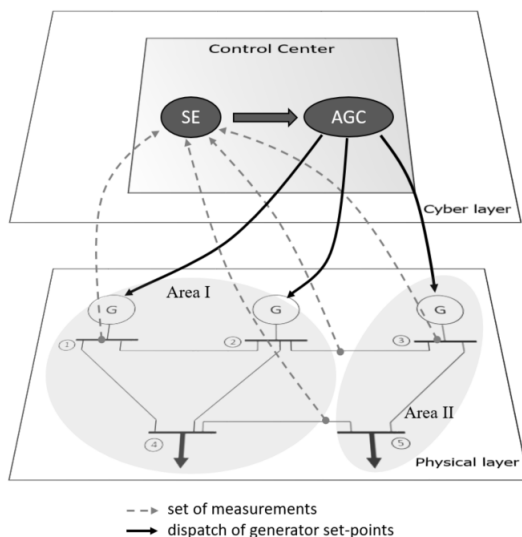
In this method, we first calculate moving average for each 10-data sequence, then convert the moving average to frequency domain and get the minimum transformed value (MTV) of each sequence. As shown in Fig. 4.7, the MTVs have significant differences. The MTV of normal moving average is around 0.0 while the ones with scale attack is around -0.2. The attacked MTVs can be separated from normal ones by setting a threshold. If a data sequence's MTV is larger than the threshold, it is normal data. Otherwise, it is regarded as attacked data. Such threshold can be set by observing the differences between MTVs of attacked data and normal data.

To localize attacks, we also perform same Fourier Transformation on tie-line power line and frequency measurements. Then we can set a threshold to check whether the measurements' converted MTV values are under attack.

## 4.4 Simulation System and Dataset

### 4.4.1 Simulation System

We simulated a 5-bus power system [38] as shown in Fig. 4.8, which is a typical power system with two interconnected control areas. The two control areas are connected



**Figure 4.8:** 5-bus system with 2 control areas

by two tie-lines. Area I contains buses 1, 2 and 4, and Area II contains buses 3 and 5. Bus 4 and Bus 5 are the load buses for Control Area I and Area II respectively. Buses 1-3 are generator buses. Area I is equipped with two generators and Area II is equipped with one generator. The power system dynamics are modeled by using the structure-preserving load model [26], [6] and the well-known generator model with governor control [25].

Each control area is equipped with its own AGC, which sits in the control center. The control center periodically collects frequency and tie-line flow measurements, then check the measurements with SE method, and then pass the measurements to AGC. AGC calculates ACE based on the measurement with Eq. (4.1) and then dispatches new set-points to generators.

#### 4.4.2 Simulation Dataset

We simulated the system with real load consumption data used at the load buses. That way, realistic ACE patterns were inserted into this synthetic AGC system. We used



real time actual load measurements and the load forecast data from two areas in NY-ISO [42], and generated load deviation values for our system by subtracting the forecast values from actual load values. We also scaled the load deviations down to fit the parameters of our small example grid. Then each one of those signals was placed on each of our load buses to simulate the load deviations. We simulated the AGC system, driven by these disturbances - deviations in loads, and generated realistic measurements of frequency, tie-line flows, as well as a realistic ACE signal. To obtain the attacked data, we inject false data into the tie-line power flow measurements and ACE will be calculated by AGC with the attacked tie-line power flow data.

#### **4.4.3 Real Dataset**

The real dataset used in the work is the ACE data from PJM (PJM Interconnection) [44], an electric regional transmission organization (RTO). The dataset includes four years' ACE data, from the year 2012 to 2015, with about 2 million data records. Each record provides the ACE value and its date and time. These datasets are normal data without any attack. To generate the attacked data, we inject false data to ACE directly.

#### **4.5 Evaluation**

In this section, we will test how the proposed methods perform on simulated dataset and real dataset. We first test the detection and localization performance of multi-feature LSTM-based method on simulated dataset. Then we will test the single-feature LSTM-based method, Fourier Transform-based method on the simulated dataset and compare them with multi-feature LSTM-based method respectively. We also test these methods on a real dataset. Since the real dataset has no frequency, tie-line power flow, real load and forecast load data except ACE data, multi-feature LSTM-based method cannot be tested on real dataset. Only

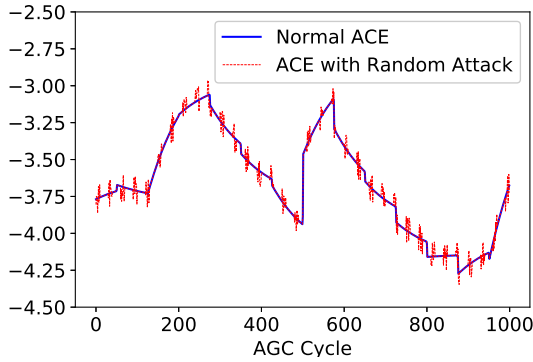
single-feature LSTM and Fourier Transform-based methods will be tested on real dataset.

#### 4.5.1 Multi-Feature LSTM-based Method on Simulated Dataset

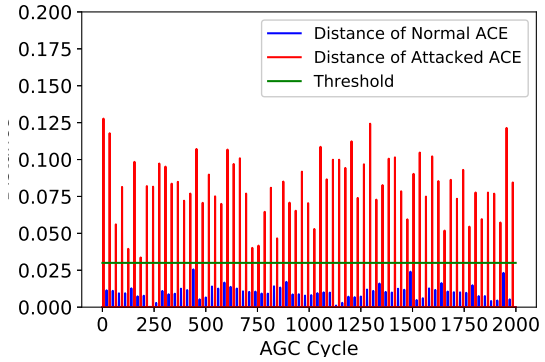
This dataset used in the experiments is the simulated dataset as described in Section 4.4.2. The simulated dataset includes about 1 million records and each record has ACE, frequency, tie-line power flow, real load and forecast load data. We split the dataset into two parts: 67% as training data and 33% as testing data. The training dataset is used to train the LSTM model, which has a hidden layer with 100 neurons and an output layer to make predictions. The sigmoid function is used as activation function for the LSTM neurons. Here  $n$  is set as 10 because the shortest attacked sequence which can negatively influence the system is 10 as discussed in [57]. The input data sequence size  $m$  is set as 5, which can be adjusted based on different power systems' datasets. The attacks are only launched periodically into testing data every 10 cycles. To test the model's performance, we feed the attacked data into the model to check the True Positive (TP) detection rate which is defined as the fraction of attacks successfully detected. We also feed the normal data without attacks into the model to see the False Positive (FP) detection rate which is defined as the fraction of normal data sequences falsely detected as attacked data. We also test the localization rate to see how many attacks can be localized correctly.

The setting of the threshold  $\theta$  is critical. If the threshold is too low, some normal data sequences will be detected as attacked data. If the threshold is too high, the attacked data sequences may not be detected. The higher is the threshold and the lower is the FP rate. In the following, we set the threshold as 0.03, which has a FP rate of less than 5%.

**Random Attack:** In this experiment, we set the attack lower bound  $\lambda_a = 0$  and launched random attacks to tie-line power flow measurements periodically. The ACEs are calculated with the attacked tie-line power flow measurements. Fig. 4.9 shows the ACE



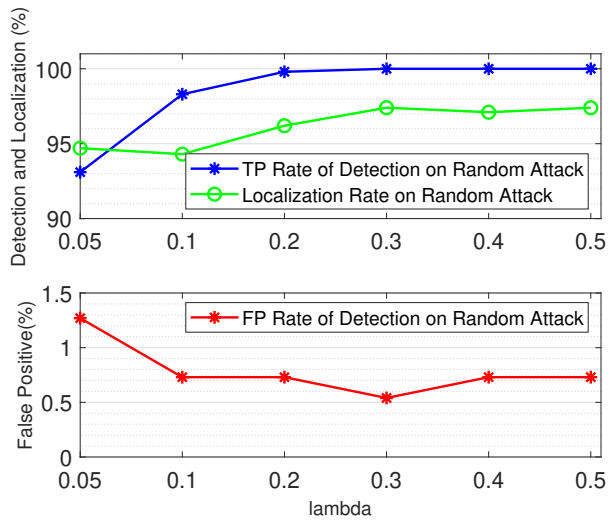
**Figure 4.9:** ACE and ACE with Random Attack



**Figure 4.10:** Distances of Normal ACE and ACE with Random Attacks

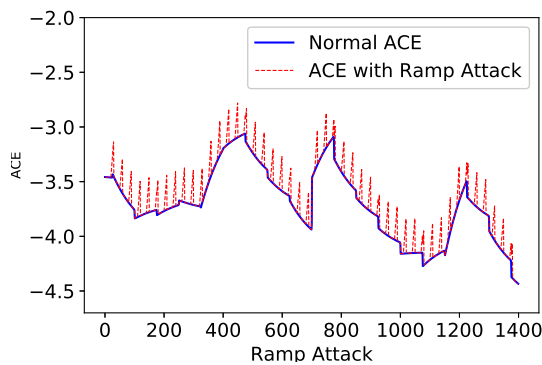
with random attacks when  $\lambda_b = 0.1$ . The red line shows ACE with random attacks and blue line show normal ACE data. As shown in this figure, the average of normal ACE is about -3.5. The random attacks with  $\lambda_b = 0.1$  will change ACE by less than 0.1, which is only about 2.8%. LSTM model will calculate the distances between predicted ACEs and real ACE measurements and then compare the distances with threshold as we discussed in Section 4.3.2. The calculated distances are shown in Fig. 4.10. The red bars show the distances between attacked ACE measurements and predicted ACEs, and blue bars show the distances between normal ACE measurements and predicted ACEs. The threshold is set as  $\theta = 0.3$ . From the figure we can see that almost all the distances of normal ACEs are under threshold while the distances of attacked ACEs are above the threshold. The detection and localization results are shown in Fig. 4.11. The results show that when  $\lambda$  is higher, the TP detection rate is also higher. This is because higher  $\lambda$  means the attacks have more significant modifications on ACE data and thus such attacks are easier to be detected (note that these attacks also have higher impact to the power grid). When  $\lambda = 0.1$ , the attack only change ACEs by less than 2.8%, but it can still detect more than 98% attacks. All the FP rates are under 1.3% and the localization rates are above 94%.

**Ramp Attack:** Fig. 4.12 shows the ACE with ramp attacks when  $\lambda = 0.04$ . On

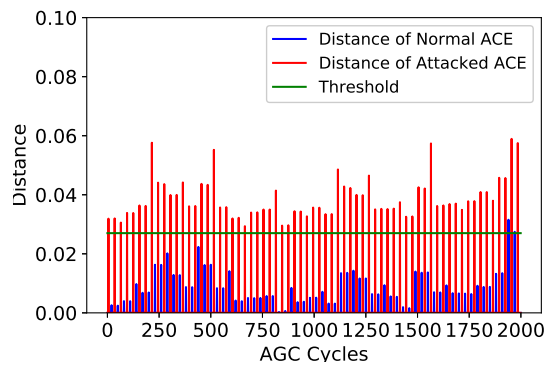


**Figure 4.11:** Detection and Localization of Multi-Feature LSTM on Random Attack

average, the attacks add about 0.2 to normal ACEs and the ramp attacks with  $\lambda = 0.04$  will change ACEs by about 5%. Similar to random attacks, Fig. 4.13 shows that almost all the distance bars of attacked ACEs are above threshold while the distances of normal ACEs are under threshold. Its detection and localization results are shown in Fig. 4.14. When  $\lambda$  is higher, the TP detection rate is also higher. When  $\lambda \geq 0.04$ , More than 97% attacks can be detected. The FP is under 3.5% and the localization is above 94%.

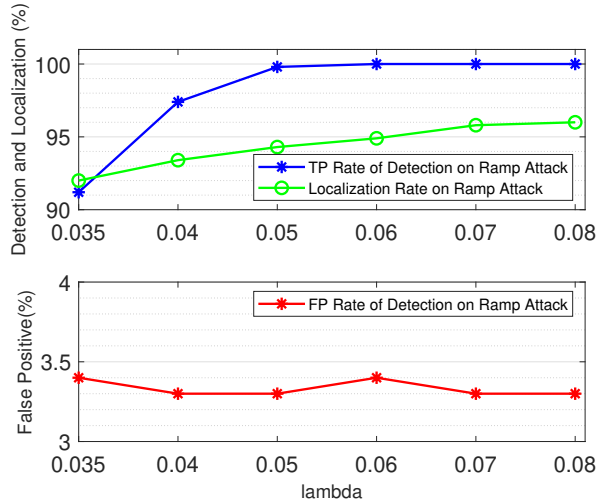


**Figure 4.12:** ACE and ACE with Ramp Attack

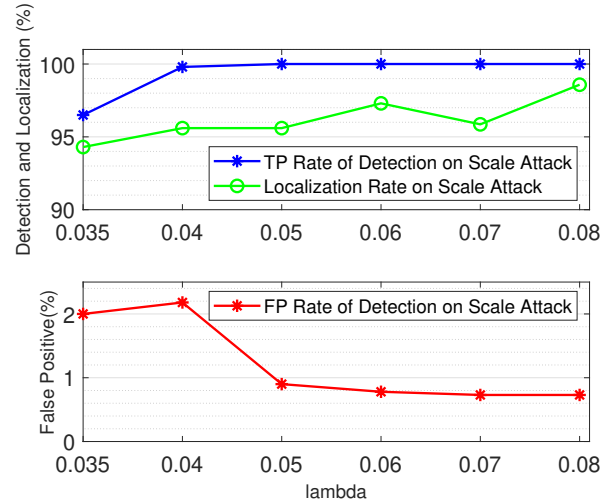


**Figure 4.13:** Distance of Normal ACE and ACE with Ramp Attacks

**Scale Attack:** Fig. 4.16 shows the ACE with scale attacks when  $\lambda = 0.04$ . From

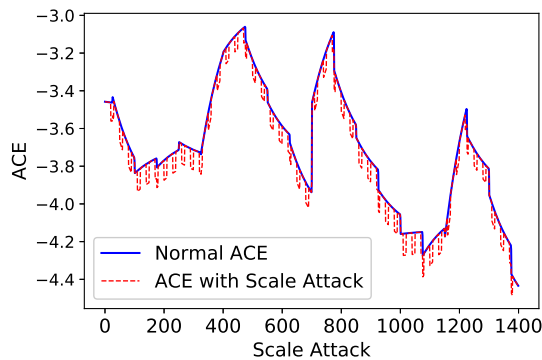


**Figure 4.14:** Detection and Localization of Multi-Feature LSTM on Ramp Attack

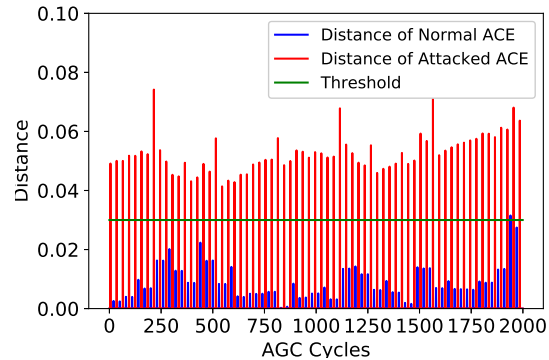


**Figure 4.15:** Detection and Localization of Multi-Feature LSTM on Scale Attack

the figure, it can be seen that the attacks change ACEs by about 4%. Similar to random attacks, Fig.4.17 shows that the threshold can separate the distances of attacked ACEs from the distances of normal ACEs very well. Its detection and localization results are shown in Fig. 4.15. When  $\lambda$  is higher, the TP detection rate is also higher. When  $\lambda \geq 0.04$ , More than 99% attacks can be detected. The FP is under 2% and the localization is above 94%.

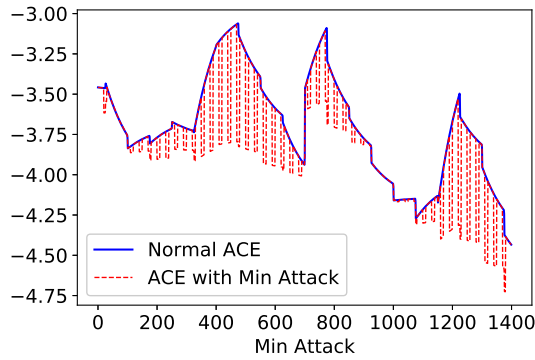


**Figure 4.16:** ACE and ACE with Scale Attack

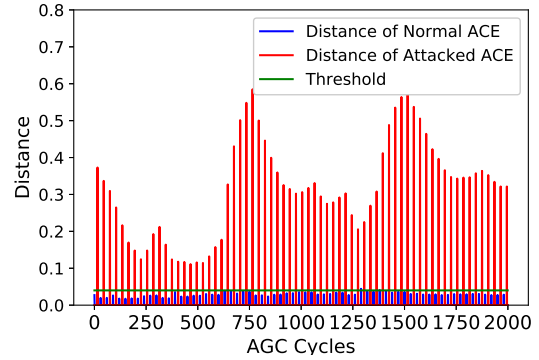


**Figure 4.17:** Distance of Normal ACE and Attacked ACE with Scale Attacks

**Min Attack:** In this experiment, we launched Min attacks by replacing real ACEs with the minimum ACEs in the last 400 cycles. Fig. 4.18 shows the ACE with Min attacks. It can be seen that in some AGC cycles, the attacks are very obvious while in some cycles

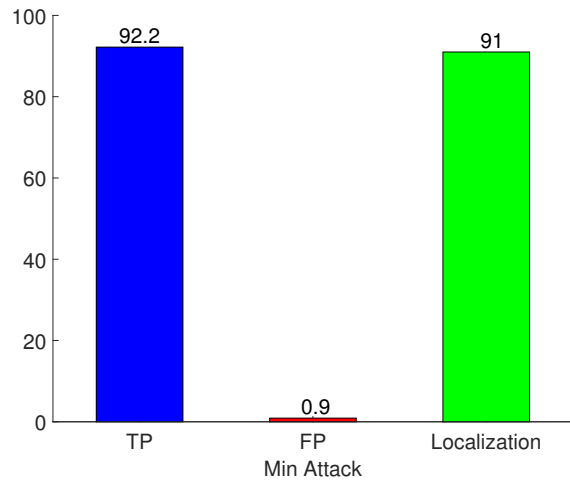


**Figure 4.18:** ACE and ACE with Min Attack

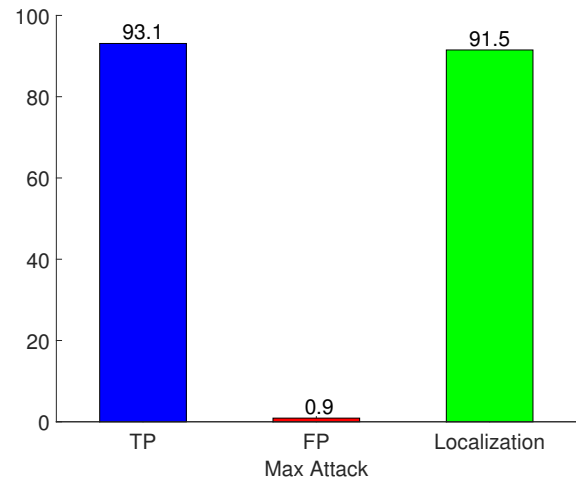


**Figure 4.19:** Distance of Normal ACE and Attacked ACE with Min Attacks

the attacks can be negligible. This is because the current ACE is the minimum ACE so far and thus the real ACE measurement is used as the minimum attacked data. The Min attack data is actually the real ACE data and has no any modification on real ACEs. When the Min attacks are the ACEs themselves, our method cannot detect them and there is also no need to detect them since they don't have any impact on the power system. That is why the TP rate shown in Fig. 4.20 is not as high as other attacks such as random attacks. The FP rate is still very low, which is about 0.9% and the localization rate is about 91%.

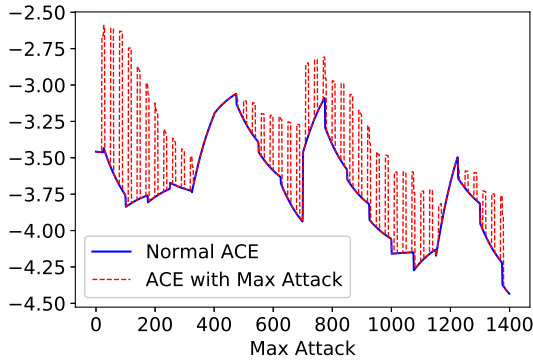


**Figure 4.20:** Detection and Localization of Multi-Feature LSTM on Min Attack

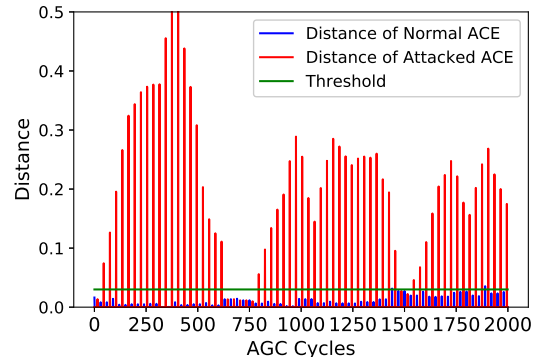


**Figure 4.21:** Detection and Localization of Multi-Feature LSTM on Max Attack

**Max Attack:** In this experiment, we launched Max attacks by replacing real ACEs



**Figure 4.22:** ACE and ACE with Max Attack



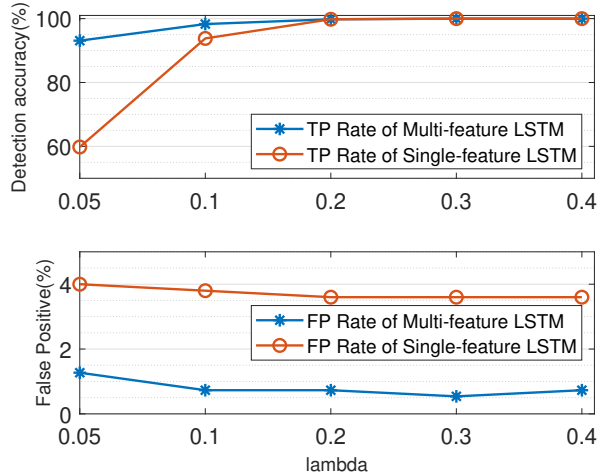
**Figure 4.23:** Distance of Normal ACE and Attacked ACE with Max Attacks

with the maximum ACEs in last 400 cycles. Fig. 4.22 shows the ACE with Max attacks. It has similar trends and attributes with Min attacks. The TP rate is shown in Fig. 4.21, which is similar to Max attacks. This is because in some AGC cycles, the maximum attack data is the real ACEs themselves. Such attacks have no modification on real ACEs and thus have no impacts on power systems.

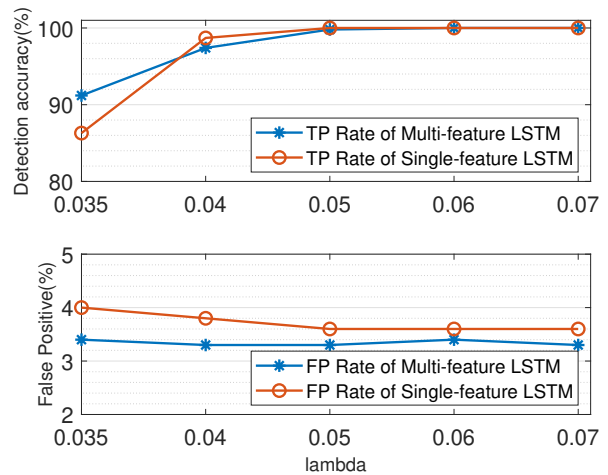
#### 4.5.2 Comparison with Single-Feature LSTM

In this section, we test the performance of single-feature LSTM-based detection method on the simulated dataset and also compare multi-feature LSTM-based method with it. Because of the space limitation, we will only compare the detection performances on random attack, ramp attack and scale attack.

**Comparison On Random Attack:** Fig. 4.24 shows the comparison between multi-feature LSTM and single-feature LSTM under same settings and same attacks. From the figure, we can see that single-feature LSTM-based method also has great performance on random attack detection. When  $\lambda = 0.1$ , it can detect more than 95% attacks. However, the multi-feature LSTM-based method still outperforms it, especially when the attack parameter is low, which means attacks are not obvious. For example, when  $\lambda = 0.05$ , the multi-feature



**Figure 4.24:** Comparison between Multi-feature LSTM and Single-feature LSTM on Random Attack



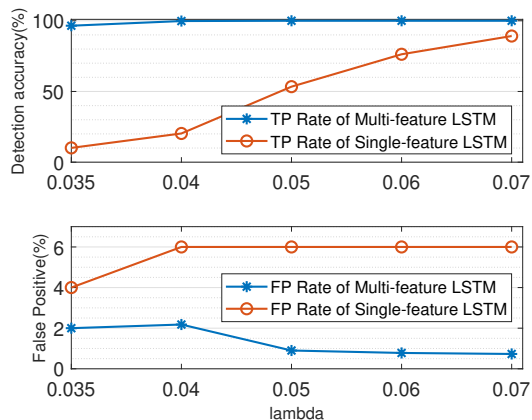
**Figure 4.25:** Comparison between Multi-feature LSTM and Single-feature LSTM on Ramp Attack

LSTM has about 93% detection rate, while single-feature LSTM has only 60%.

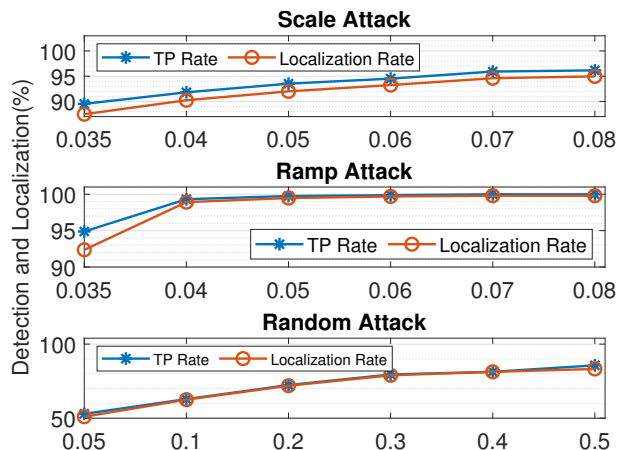
**Comparison On Ramp Attack:** Fig. 4.25 shows the comparison between multi-feature LSTM and single-feature LSTM under same settings and ramp attacks. The results show that single-feature LSTM-based method has great performance on ramp attack detection and the multi-feature LSTM-based method still outperforms it.

**Comparison On Scale Attack:** Fig. 4.26 shows the comparison between multi-feature LSTM and single-feature LSTM under scale attacks. It can be seen that the single-feature LSTM-based method performs poorly on scale attack detection. When  $\lambda = 0.05$ , it can only detect about 50% attacks. This is because LSTM model learns data patterns and then detects attacks based on predicted patterns. However, scale attacks just scale the data's value up or down and do not change the ACE data sequence patterns. Single-feature LSTM-based method will assume ACE is normal if its pattern is normal. Different from single-feature LSTM which only relies on ACE data, multi-feature LSTM also relies on many other data such as frequency and real load in addition to ACEs. Even though ACE pattern is not changed, it can still be found whether ACE value is normal with the help of



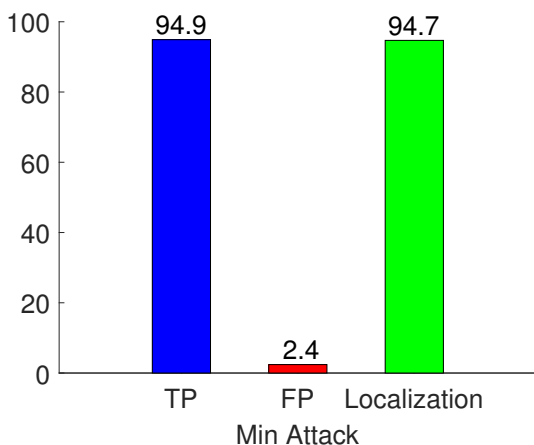


**Figure 4.26:** Comparison between Multi-feature LSTM and Single-feature LSTM on Scale Attack

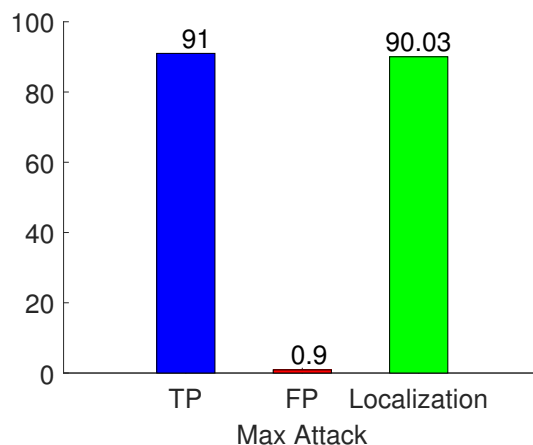


**Figure 4.27:** Detection and Localization of Fourier Transform

other measurement data.



**Figure 4.28:** Fourier Transform on Min Attack



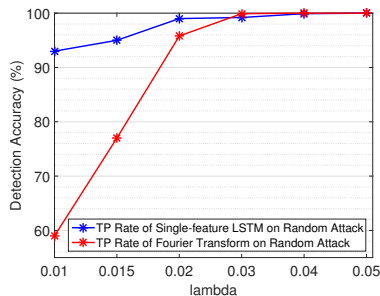
**Figure 4.29:** Fourier Transform on Max Attack

### 4.5.3 Fourier Transform-based Method on Simulated Dataset

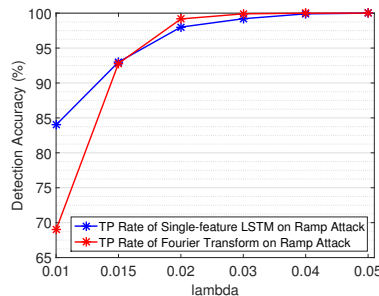
In this section, we test how fourier transform-based method performs on attack detection and localization. We launched same attacks with multi-feature LSTM testing. Fig. 4.27 shows the detection and localization accuracy of Fourier Transform-based method under scale, ramp and random attacks. The threshold is set as 0.03 when the FP rate is 4.3%. As shown in Fig. 4.27, Fourier Transform method can have more than 90% detection and

localization accuracy for scale and ramp attacks, but it does not perform very well in random attack detection and localization. The detection and localization on Min and Max attacks are shown in Fig. 4.28 and Fig. 4.29 respectively.

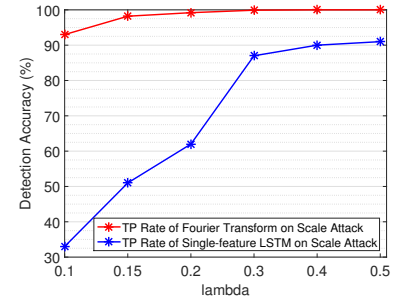
#### 4.5.4 LSTM and Fourier Transform-based Methods on Real Dataset



**Figure 4.30:** Detection of Single-Feature LSTM and Fourier Transform on Random Attack



**Figure 4.31:** Detection of Single-Feature LSTM and Fourier Transform on Ramp Attack



**Figure 4.32:** Detection of Single-Feature LSTM and Fourier Transform on Scale Attack

In this section, we test the LSTM-based method and Fourier Transform-based method on the real dataset. Since there is no frequency, tie-line power flow, real load and forecast load data in the real dataset, we are not able to test multi-feature LSTM-based method on the real dataset. Single-feature LSTM and Fourier Transform-based method are tested. As we discussed in Section 4.2.2, same attack parameters may have different impacts on different power system. For this power system where the real dataset was generated, when random attack  $\lambda_b = 0.1$ , ramp attack  $\lambda_r = 0.1$ , and scale attack parameter  $\lambda_s = 0.1$ , ACE are changed by about 5%, 5%, and 10% on average, respectively. The detection results of single-feature LSTM and Fourier Transform on random attack, ramp attack and scale attack are shown in Fig. 4.30, Fig. 4.31 and Fig. 4.32. The results show that the single-feature LSTM has better performance than Fourier Transform on random and ramp attacks. For example, when random attack's parameter  $\lambda = 0.1$ , single-feature LSTM can detect more

than 90% random attacks. However, it performs poorly on scale attack as we discussed in Section 4.5.2. The Fourier Transform is able to detect scale attacks effectively as shown in Fig. 4.32. Therefore, we can use single-feature LSTM-based method to detect random and ramp attacks and use Fourier Transform as the complementary method to detect scale attacks.

## 4.6 Related Work

Some work have been done about attacks on AGC. In [57], the work explored how to launch attacks to achieve expected effects in the shortest time, but no detection method was given. In [51], Sridhar et al. developed a model-based anomaly detection algorithm, in which the ACE values were predicted in 5-minute intervals based on load forecast. The real-time value of ACE will be regarded as an anomaly if it is not in the forecast range. This method heavily depends on load forecast. However, in the practical power system, load forecasting accuracy is not high enough, which will affect the ACE prediction accuracy. The work in [2] presents a two-tier intrusion detection system. The first tier forecasts the ACE value for the next time instance based on the current time instance. The measurement deviating from the prediction will be flagged as anomalous and then the flagged instance is passed to the second tier to verify anomaly by incorporating the overall system variable. However, the algorithm only uses the data of one time point and if this time point's data is abnormal or attacked, its prediction will be misled. The approach presented in [34] adopts a security game model to choose the best response strategies against attackers, but it does not give an approach to detect the attacks.

The work in [31] proposed physics-based and learning-based methods to detect false data injection attacks. The physics-based method uses alternative computation of ACE by using a more detailed model of the control area and the learning-based method uses

single-feature LSTM model to predict ACEs. However, the physics-based method requires more noise-free measurements and needs more complex computations, and the learning-based method cannot detect scale attacks effectively. In addition, they are not tested on real datasets and it is unknown how they will perform in real world. [20] develops a deep learning-based method, Conditional Deep Belief Network, to detect false data injection attacks. It requires attacked data as the training dataset and thus how the model performs heavily depends on what types of attacks are included in the training dataset. If one attack type is not included in the training dataset, it will be difficult to be detected. However, there are not many available attacked data that can be used as training data in practice.

#### 4.7 Summary

In this chapter we mainly propose Neural Network-based method (multi-feature LSTM and single-feature LSTM) and Fourier Transform-based method to detect and localize false data injection attacks. We test these methods on random attack, ramp attack, scale attack, min attack and max attack on real and simulated datasets. The experiments show both LSTM-based and Fourier transform-based methods have promising performance on attack detection and localization. Specifically, multi-feature LSTM-based has better performance than single-feature LSTM and Fourier transform-based method. The single-feature LSTM-based method can detect most of the attacks but cannot detect scale attacks effectively, while Fourier Transform-based method has good performance on scale attacks. They can be used as complementary detection methods.

## 5 Conclusions and Future Work

### 5.1 Conclusions

In this dissertation, we studied how to help counter the vulnerabilities in power systems. Specifically, we automatically analyzed the vulnerabilities and determined remediation actions, optimized remediation action applying order to reduce the system's security risk, and identified whether there is any vulnerability that has been exploited for this system.

We first proposed a machine learning-based framework to automate remediation decision analysis for electric utilities in Chapter 2, which applied a predictive decision tree model over vulnerability features and asset features to predict the remediation decision for each vulnerability. The model is built over historical, manual remediation decision data to capture and mimic how human operators make decisions, but it can make decisions much more quickly than manual analysis. We tested an instance of the framework customized for one electric utility over two datasets obtained from the utility. Results showed high prediction accuracy and time savings.

We then scheduled the remediation action applying order so that the system's total risk is as low as possible in Chapter 3. We predicted the vulnerabilities' probability of being exploited over time. Each vulnerability's risk can be calculated based on its impact score on the system and the predicted probability. Then we model the scheduling problem as MIP model and get the optimized scheduling patching order so that the total risk is minimum. The evaluation based on a utility company's real data shows our method can significantly reduce the risk. This scheduling optimization is the following step of the work in Chapter 2. After we determine the vulnerabilities' remediation actions, we can first apply the optimization method on the group of vulnerabilities that need to be patched and mitigated, to determine the patching and mitigation order. Then apply this method on the vulnerabilities that can

be patched later and get their optimized scheduling order.

In Chapter 4, we identify whether a vulnerability has already been exploited specifically for a power system. We give a detailed case study about how to identify exploited vulnerabilities by detecting attacks in the AGC system. We proposed Neural Network-based method (multi-feature LSTM and single-feature LSTM) and Fourier Transform-based method to detect false data injection attacks. We test these methods on random attack, ramp attack, scale attack, min attack and max attack with real and simulated datasets. The experiments show that both LSTM-based and Fourier Transform-based method have good performance and can also be used as complementary detection methods.

## 5.2 Future Work

In the dissertation, we studied how to counter vulnerabilities in power systems. There is still more work to explore.

- Explore better-than-human remediation decisions: in Chapter 2 we applied machine learning techniques to make remediation decisions by mimicking human reasoning. In fact, machine learning can take much more factors and address more complicated rules than human and thus can help make better decisions. We will explore how to make better-than-human remediation decisions by utilizing machine learning techniques.
- Explore how to apply remediation actions with less human intervention: after scheduling the remediation actions applying order, operators still need manually apply the remediation actions. In the future work, we will analyze the patch and devices/software characteristics, and patch workflow to automate the remediation applying process.
- Identify more exploited vulnerabilities: in this dissertation, we mainly identified exploited vulnerabilities by detecting false data injection attacks. In the future, we will

identify more exploited vulnerabilities by monitoring and analyzing the devices and communication network activities.

## Bibliography

- [1] Jacob A. Harer et al. “Automated software vulnerability detection with machine learning”. In: (Feb. 2018).
- [2] Muhammad Qasim Ali et al. “Two-tier data-driven intrusion detection for automatic generation control in smart grid”. In: *Communications and Network Security (CNS), 2014 IEEE Conference on*. IEEE. 2014, pp. 292–300.
- [3] Luca Allodi and Fabio Massacci. “Attack Potential in Impact and Complexity”. In: *Proceedings of the 12th International Conference on Availability, Reliability and Security*. ACM. 2017, p. 32.
- [4] Ali Alshawish and Hermann de Meer. “Risk-based Decision-Support for Vulnerability Remediation in Electric Power Networks”. In: *Proceedings of the Tenth ACM International Conference on Future Energy Systems*. ACM. 2019, pp. 378–380.
- [5] Ashish Arora et al. “An empirical analysis of software vendors’ patch release behavior: impact of vulnerability disclosure”. In: *Information Systems Research* 21.1 (2010), pp. 115–132.
- [6] Arthur R Bergen and David J Hill. “A structure preserving model for power system stability analysis”. In: *IEEE Transactions on Power Apparatus and Systems* 1 (1981), pp. 25–35.
- [7] E Oran Brigham and RE Morrow. “The fast Fourier transform”. In: *IEEE spectrum* 4.12 (1967), pp. 63–70.
- [8] North American Electric Reliability Corporation. *CIP Standards*. <https://www.nerc.com/pa/Stand/Pages/CIPStandards.aspx>.
- [9] Zoho Corporation. *ManageEngine Patch Manager Plus*. <https://www.manageengine.com/patch-management/?itsecuritySol>.
- [10] CVSS. URL: <https://www.first.org/cvss/>.
- [11] Exploit Database. <https://www.exploit-db.com/>.
- [12] Doble Engineering. *Doble PatchAssure*. <https://www.doble.com/product/patchassure/>.
- [13] Flexera. <https://www.flexera.com/producer/>.
- [14] Stefan Frei et al. “Large-scale vulnerability analysis”. In: *Proceedings of the 2006 SIGCOMM workshop on Large-scale attack defense*. ACM. 2006, pp. 131–138.
- [15] Christian Fruhwirth and Tomi Mannisto. “Improving CVSS-based vulnerability prioritization and response with context information”. In: *Proceedings of the 2009 3rd international Symposium on Empirical Software Engineering and Measurement*. IEEE Computer Society. 2009, pp. 535–544.



- [16] Seyed Mohammad Ghaffarian and Hamid Reza Shahriari. “Software vulnerability analysis and discovery using machine-learning and data-mining techniques: A survey”. In: *ACM Computing Surveys (CSUR)* 50.4 (2017), p. 56.
- [17] Gabriele Gianini et al. “A game theoretic approach to vulnerability patching”. In: *2015 International Conference on Information and Communication Technology Research (Ictrc)*. IEEE. 2015, pp. 88–91.
- [18] Antonio Gomez-Exposito and Ali Abur. *Power system state estimation: theory and implementation*. CRC press, 2004.
- [19] Gustavo Grieco et al. “Toward large-scale vulnerability discovery using machine learning”. In: *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy*. ACM. 2016, pp. 85–96.
- [20] Youbiao He, Gihan J Mendis, and Jin Wei. “Real-time detection of false data injection attacks in smart grid: A deep learning-based intelligent mechanism”. In: *IEEE Transactions on Smart Grid* 8.5 (2017), pp. 2505–2516.
- [21] Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory”. In: *Neural computation* 9.8 (1997), pp. 1735–1780.
- [22] Department of Homeland Security. *Recommended Practice for Patch Management of Control Systems*. [https://ics-cert.us-cert.gov/sites/default/files/recommended\\_practices/RP\\_Patch\\_Management\\_S508C.pdf](https://ics-cert.us-cert.gov/sites/default/files/recommended_practices/RP_Patch_Management_S508C.pdf). 2008.
- [23] Jin B Hong, Dong Seong Kim, and Abdelkrim Haqiq. “What vulnerability do we need to patch first?” In: *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE. 2014, pp. 684–689.
- [24] Yu-Lun Huang et al. “Understanding the physical and economic consequences of attacks on control systems”. In: *International Journal of Critical Infrastructure Protection* 2.3 (2009), pp. 73–83.
- [25] Marija D Ilic and John Zaborszky. *Dynamics and control of large electric power systems*. Wiley New York, 2000.
- [26] Marija D Ilic et al. “Modeling of future cyber–physical energy systems for distributed sensing and control”. In: *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans* 40.4 (2010), pp. 825–838.
- [27] *Impact Score*. URL: <https://www.first.org/cvss/v2/guide>.
- [28] Forum of Incident Response and Security Teams. *CVSS*. ”<https://www.first.org/cvss/v2/guide>”.
- [29] Ponemon Institute. *Today’s State of Vulnerability Response: Patch Work Demands Attention*. <https://www.servicenow.com/content/dam/servicenow/documents/analyst-research/ponemon-state-of-vulnerability-response.pdf>. 2018.

- [30] Nasser Jaleeli et al. “Understanding automatic generation control”. In: *IEEE transactions on power systems* 7.3 (1992), pp. 1106–1122.
- [31] Ana Jevtic et al. “Physics-and Learning-based Detection and Localization of False Data Injections in Automatic Generation Control”. In: *IFAC-PapersOnLine* 51.28 (2018), pp. 702–707.
- [32] HyunChul Joh and Yashwant K Malaiya. “Defining and assessing quantitative security risk measures using vulnerability lifecycle and cvss metrics”. In: *The 2011 international conference on security and management (sam)*. 2011, pp. 10–16.
- [33] Prabin B Lamichhane, Liang Hong, and Sachin Shetty. “A Quantitative Risk Analysis Model and Simulation Of Enterprise Networks”. In: *2018 IEEE 9th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*. IEEE. 2018, pp. 844–850.
- [34] Yee Wei Law, Tansu Alpcan, and Marimuthu Palaniswami. “Security games for risk minimization in automatic generation control”. In: *IEEE Transactions on Power Systems* 30.1 (2015), pp. 223–232.
- [35] Tao Lei, Regina Barzilay, and Tommi Jaakkola. “Rationalizing neural predictions”. In: *arXiv preprint arXiv:1606.04155* (2016).
- [36] Frank Li and Vern Paxson. “A Large-Scale Empirical Study of Security Patches”. In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM. 2017, pp. 2201–2215.
- [37] Shu-Hsien Liao. “Expert system methodologies and applications—a decade review from 1995 to 2004”. In: *Expert systems with applications* 28.1 (2005), pp. 93–103.
- [38] Xiaojun Zhang Liu. “Structural modeling and hierarchical control of large-scale electric power systems”. PhD thesis. Massachusetts Institute of Technology, 1994.
- [39] Louai Maghrabi et al. “Improved software vulnerability patching techniques using CVSS and game theory”. In: *2017 International Conference on Cyber Security And Protection Of Digital Services (Cyber Security)*. IEEE. 2017, pp. 1–6.
- [40] Danilo P Mandic and Jonathon Chambers. *Recurrent neural networks for prediction: learning algorithms, architectures and stability*. John Wiley & Sons, Inc., 2001.
- [41] Antonio Nappa et al. “The attack of the clones: A study of the impact of shared code on vulnerability patching”. In: *Security and Privacy (SP), 2015 IEEE Symposium on*. IEEE. 2015, pp. 692–708.
- [42] *Real-time actual load data reports*. 2018. URL: <http://www.nyiso.com/public/markets%20operations>.
- [43] *OR-Tools*. URL: <https://developers.google.com/optimization>.

- [44] *PJM ACE data*. 2017. URL: <http://www.pjm.com/markets-and-operations/etools/oasis/system-information/historical-area-control-error-data.aspx>.
- [45] Rebecca Russell et al. “Automated vulnerability detection in source code using deep representation learning”. In: *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE. 2018, pp. 757–762.
- [46] Muhammad Shahzad, Muhammad Zubair Shafiq, and Alex X Liu. “A large scale exploratory analysis of software vulnerability life cycles”. In: *Proceedings of the 34th International Conference on Software Engineering*. IEEE Press. 2012, pp. 771–781.
- [47] Anoop Singhal and Xinming Ou. “Security risk analysis of enterprise networks using probabilistic attack graphs”. In: *Network Security Metrics*. Springer, 2017, pp. 53–73.
- [48] GFI Software. *GFI LanGuard*. <https://www.gfi.com/products-and-solutions/network-security-solutions/gfi-languard>.
- [49] Solarwinds. *Patch Manager*. <https://www.solarwinds.com/patch-manager>.
- [50] Foxguard Solutions. <https://foxguardsolutions.com/>.
- [51] Siddharth Sridhar and Manimaran Govindarasu. “Model-based attack detection and mitigation for automatic generation control”. In: *IEEE Transactions on Smart Grid* 5.2 (2014), pp. 580–591.
- [52] U.S. National Institute of Standards and Technology. <https://nvd.nist.gov/vuln/full-listing>.
- [53] U.S. National Institute of Standards and Technology. *CPE*. <https://nvd.nist.gov/products/cpe>.
- [54] U.S. National Institute of Standards and Technology. *National Vulnerability Database*. <https://nvd.nist.gov/vuln>.
- [55] U.S. National Institute of Standards and Technology. *NVD Data Feeds*. <https://nvd.nist.gov/vuln/data-feeds>.
- [56] Gary Stoneburner, Alice Goguen, and Alexis Feringa. “Risk Management Guide for Information Technology Systems”. In: *NIST Special Publication ()*, pp. 800–30.
- [57] Rui Tan et al. “Optimal false data injection attack against automatic generation control in power grids”. In: *Proceedings of the 7th International Conference on Cyber-Physical Systems*. IEEE Press. 2016, p. 2.
- [58] Tenable. *Tenable Predictive Prioritization*. <https://www.tenable.com/cyber-exposure/platform>.
- [59] *TF-IDF*. URL: <http://www.tfidf.com/>.

- [60] Sirikwan Treetippayaruk and Twittie Senivongse. “Security vulnerability assessment for software version upgrade”. In: *Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), 2017 18th IEEE/ACIS International Conference on*. IEEE. 2017, pp. 283–289.
- [61] *Vulners Database*. URL: <https://vulners.com/>.
- [62] Chaowei Xiao and Armin Sarabi. “From Patching Delays to Infection Symptoms: Using Risk Profiles for an Early Discovery of Vulnerabilities Exploited in the Wild”. In: *SEC’18 Proceedings of the 27th USENIX Conference on Security Symposium*. USENIX. 2018, pp. 903–918.
- [63] Le Xie, Yilin Mo, and Bruno Sinopoli. “Integrity data attacks in power market operations”. In: *IEEE Transactions on Smart Grid* 2.4 (2011), pp. 659–666.
- [64] Fabian Yamaguchi, Felix Lindner, and Konrad Rieck. “Vulnerability extrapolation: assisted discovery of vulnerabilities using machine learning”. In: *Proceedings of the 5th USENIX conference on Offensive technologies*. USENIX Association. 2011, pp. 13–13.
- [65] Fengli Zhang and Qinghua Li. “Deep learning-based data forgery detection in automatic generation control”. In: *Communications and Network Security (CNS):International Workshop on Cyber-Physical Systems Security (CPS-Sec), 2017 IEEE Conference on*. IEEE. 2017, pp. 400–404.
- [66] Fengli Zhang and Qinghua Li. “Security Vulnerability and Patch Management in Electric Utilities: A Data-Driven Analysis”. In: *Proceedings of the First Workshop on Radical and Experiential Security*. ACM. 2018, pp. 65–68.
- [67] Maede Zolanvari et al. “Machine learning based network vulnerability analysis of industrial Internet of Things”. In: *IEEE Internet of Things Journal* (2019).