A Machine Learning-based Approach for Automated Vulnerability Remediation Analysis

Fengli Zhang*, Philip Huff**, Kylie McClanahan*, Qinghua Li*
*University of Arkansas, Fayetteville, AR USA, {*fz002, klmcclan, qinghual*}@uark.edu
**University of Arkansas, Little Rock, AR USA, *pdhuff*@ualr.edu

***Abstract*—Security vulnerabilities in firmware/software pose an important threat to power grid security, and thus electric utility companies should quickly decide how to remediate vulnerabilities after they are discovered. Making remediation decisions is a challenging task in the electric industry due to the many factors to consider, the balance to maintain between patching and service reliability, and the large amount of vulnerabilities to deal with. Unfortunately, remediation decisions are current manually made which take a long time. This increases security risks and incurs high cost of vulnerability management. In this paper, we propose a machine learning-based automation framework to automate remediation decision analysis for electric utilities. We apply it to an electric utility and conducts extensive experiments over two real operation datasets obtained from the utility. Results show the high effectiveness of the solution.**

***Index Terms*—vulnerability and patch management, power system security, machine learning**

I. Introduction

Vulnerabilities in software/firmware 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. For this reason, security vulnerability and patch management (VPM) is an integral and currently one of the most important components of power grid security [1]. Every electric utility (a company that generates, transmits and distributes electricity) has a VPM mechanism deployed at its security operations center. When security vulnerabilities with their assets are discovered, an electric utility needs to decide how to remediate the vulnerabilities quickly to reduce security risks.

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 applicable security vulnerabilities at any given time often exceeds the capacity of organization's to apply risk analysis. Over the past two 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 [2]. 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. *It is worthy to note that our machine learning approach only recommends remediation decisions and human operators have the ultimate authority to accept the predicted decisions or not.*

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 [3] 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

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

improved over the past few years [2].

The contributions of the paper are summarized as follows:

- To the best of our knowledge, this is the first work studying the feasibility of automating remediation decision analysis in VPM, which is currently manually done at electric utilities and many other organizations.
- We propose a machine learning-based framework for automating remediation action analysis based on vulnerability features and assets features in the operation environment. We choose decision tree as the learning model since it resembles human reasoning and enables generation of reason code for operators to verify the predictions if needed. We also design methods to simplify the reason codes for easier verification, and propose a group-based asset management method to simplify asset feature maintenance.
- We instantiate the framework for one electric utility, and perform extensive experimental evaluations based on two one-year datasets obtained from the electric utility. Evaluation results show that the approach is very effective.

The remainder of the paper is organized as follows. Section II reviews related work. Section III introduces current practices of VPM in electric utilities and presents results of a survey. Section IV presents the machine learning-based automation framework. Section V introduces the instantiation of the framework in one electric utility. Section VI presents evaluation results. The last two sections present discussions and conclude the paper.

II. Related Work

There are many VPM solutions available for corporate use, such as GFI LanGuard [4], Patch Manager Plus by ManageEngine [5], and Patch Manager by SolarWinds [6]. However, these solutions focus on vulnerability discovery and deployment of patches rather than the decisions necessary to optimize resources for vulnerability remediation. Most solutions designed for VPM in electric utilities also fall into this category, such as Doble Engineering's PatchAssure [7], Flexera [8], or FoxGuard Solutions [9]. They are unable to analyze vulnerabilities against the operating environment and make decisions on how to address vulnerabilities.

When addressing vulnerabilities, there are some publicly-available sources of information. The NIST NVD [10] provides a well-structured, reliable data feed of vulnerabilities and their corresponding information as they are reported. Vulners [11] has a freely-accessible API to search vulnerability information and discover available exploits; the Exploit Database [12] also allows users to search for available exploits. Tenable [13] 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. [14] and [15] analyze large vulnerability datasets and report trends in vulnerability attributes, and disclosure and discovery dates.

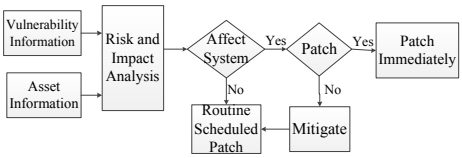

Fig. 1: Vulnerability and patch management process

[16], [17], and [18] analyze patches and patch behavior, such as the time window between when patches are released and when they are installed or effective in protecting against vulnerabilities. [19], [20], and [21] describe method for prioritizing patching based on the severity of an exploit. [22] and [23] 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 [24]–[29], but our work uses vulnerability features with asset information to determine how to remediate vulnerabilities.

III. Current Practice of VPM in Electric Utilities

This section presents the current practice for organizations in the electric sector. As recommended practice for VPM by the U.S. Department of Homeland Security (DHS) [30], which is shown in Fig. 1, when vulnerabilities are discovered, organizations first need to analyze whether a vulnerability will affect their systems by considering 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 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.

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.

IV. Machine Learning-based Framework for Remediation Action Analysis

Security operators consider many factors to decide remediation actions for vulnerabilities. The factors include vulner-

ability 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 [31] 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 (denoted as Patch-Later). If a vulnerability can be exploited, and it is at a user workstation, the decision is to patch immediately (denoted by Patch-Now); 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 (denoted by Mitigate-Now-Patch-Later).

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 [32]. 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.

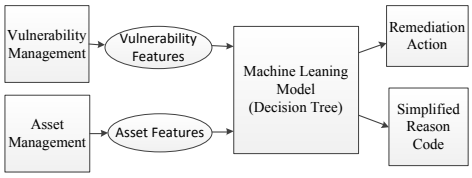

Fig. 2: Machine learning-based framework

We adopt an approach that uses machine learning to automate the analysis. We propose a machine learning-based framework (see Fig. 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 very 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 III 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 III, 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 VI-H for analysis). We acknowledge that manual decisions might not be optimal or correct, and leave it for future work to study how to make better decisions than human operators.

TABLE I: 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	

A. 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 I. 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 [31].

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 [10] 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 [33] 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.

B. 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 an asset is set as "High", loss of confidentiality will have severe impact on the asset.
- Integrity Requirement: Similar to Confidentiality Requirement but focused on integrity.
- Availability Requirement: Similar to Confidentiality Requirement but focused on availability.

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.

C. 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.

For illustration purposes, Fig. 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

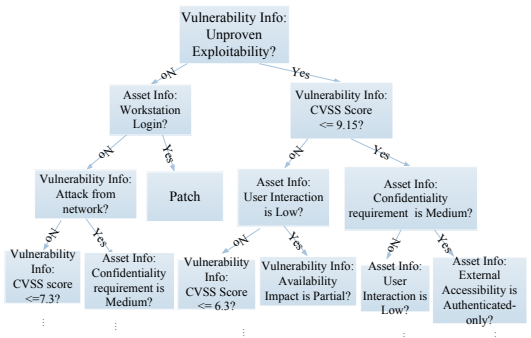

Fig. 3: An example of trained decision tree model

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.

D. 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".

E. 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 VI).

V. 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 features 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. All the features except CVSS score are categorical. We convert these categorical values to binary

data with one hot encoding. The CVSS score is normalized by scaling between 0 and 1. Gini is used as the metric to measure the split of the tree. It is worthy to note that some decision tree parameters should be tuned based on the dataset to achieve best performance (see Section VI-B 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 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 II 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 this choice is that the vulnerability is not exploitable, the CVSS score is less than 4.2 which suggests a low asset impact, and 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.

TABLE II: 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

VI. Evaluations

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

A. Dataset

We collected two datasets from Org_A, 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 features, its associated asset, and asset features. Additionally, each record also includes the remediation decision for the vulnerability made by human operators.

B. Parameter Tuning for Decision Tree

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.

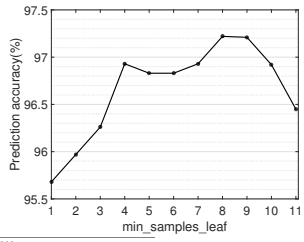

Fig. 4: Prediction over min_leaf_samples

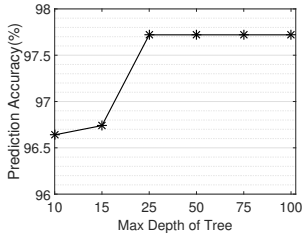

Fig. 5: Prediction over tree depth

We experimented on different minimum numbers of samples in leaf nodes and the results are shown in Fig. 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 the tree is. As shown in Fig. 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 new samples. If "min_samples_leaf" is too large where the tree is short, the prediction accuracy also decreases because the 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. 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.

C. Prediction Accuracy

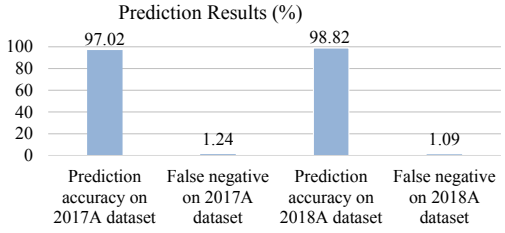

Fig. 6: Prediction accuracy on 2017A and 2018A dataset

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. 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 prediction 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. A utility company might have multiple 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 month and this month). 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.

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 for the 2018A dataset. 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.

D. 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 VI-C, the average length of reason code is 6.9 conditions. After applying the length reduction rules proposed in Section IV-D, 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. 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.

E. 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 IV-E. In this experiment, we test the model's prediction accuracy with dynamic training. In particular, we assume the operator

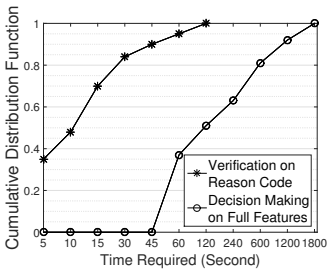

Fig. 7: Time of reason code-based and feature-based verification

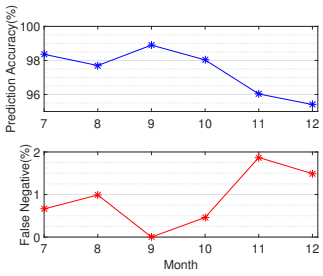

Fig. 8: Monthly prediction accuracy

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. 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.

F. Prediction with Different Feature Sets

The above experiments used 16 features including software name, vulnerability characteristics, asset name, and asset features. Here, we evaluate the usefulness of different features for prediction, and show the results over the 2017A dataset in Fig. 9. Without software name and asset name as features, the prediction accuracy only slightly decreases. However, without software name, asset name and asset features (i.e., only vulnerability characteristics), the prediction accuracy drops to 83.78%; without software name, asset name and vulnerability features (i.e., only asset features), the prediction accuracy drops to 68.33%. The results indicate that both vulnerability features and asset features are important.

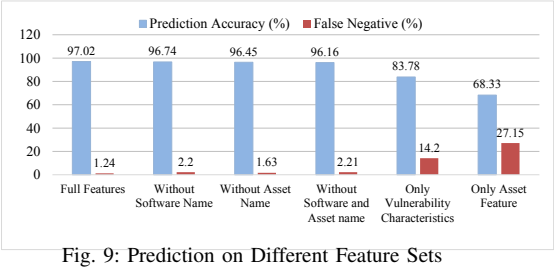

Fig. 9: Prediction on Different Feature Sets

G. Comparison with Other Models

We compare the decision tree model 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. As shown in Fig. 10, the decision tree model

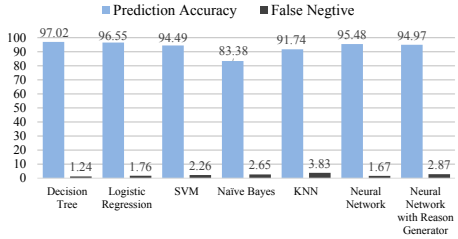

Fig. 10: Comparison with other machine learning models

performs better than other models. Logistic regression and neural network perform close to decision tree. However, they are not as easily explainable as decision tree.

Considering recent work on rationalizing neural networks, we adapted and implemented a neural network model with reason generator [34] to have a comprehensive comparison with decision tree. The details of adaptation and implementation are omitted due to the space limitation and will provided in an extended version of this paper. As shown in Fig. 10, decision tree also outperforms the neural network with reason generator in prediction. As to reason code, the average length of reason codes generated by decision tree is about 4, while that of neural network is around 8.5. When reason codes are sufficient to support predictions (see Sect. VI-D), the shorter reason codes of decision tree are easier to interpret/verify.

H. 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 VI-E. The average time of prediction verification is 28.8 seconds as shown in Section VI-D. 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 VI-D, 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, may take a rest now and then, need to perform many other duties such as meetings, reporting and training, and can be distracted by other things like chatting with each other. 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 III, 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_A. 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.

VII. 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 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.

VIII. Conclusions

This paper 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.

References

[1] "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.

[2] *https://nvd.nist.gov/vuln/data-feeds*.

[3] "Cip standards," *https://www.nerc.com/pa/Stand/Pages/CIPStandards.aspx*.

[4] "Gfi languard," *https://www.gfi.com/products-and-solutions/network-security-solutions/gfi-languard*.

[5] *https://www.manageengine.com/patch-management/?itsecuritySol*.

[6] *https://www.solarwinds.com/patch-manager*.

[7] "Doble patchassure," *https://www.doble.com/product/patchassure/*.

[8] "Flexera," *https://www.flexera.com/producer/*.

[9] "Foxguard solutions," *https://foxguardsolutions.com/*.

[10] "National vulnerability database," *https://nvd.nist.gov/vuln*.

[11] Vulners, "Vulners api," *https://vulners.com/products*.

[12] "Exploit database," *https://www.exploit-db.com/*.

[13] *https://www.tenable.com/cyber-exposure/platform*.

[14] M. Shahzad, M. Z. Shafiq, and A. X. Liu, "A large scale exploratory analysis of software vulnerability life cycles," in *Proc. of the Int'l Conference on Software Engineering*. IEEE Press, 2012, pp. 771–781.

[15] S. Frei, M. May, U. Fiedler, and B. Plattner, "Large-scale vulnerability analysis," in *ACM SIGCOMM workshop on Large-scale attack defense*, 2006, pp. 131–138.

[16] F. Li and V. Paxson, "A large-scale empirical study of security patches," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2017, pp. 2201–2215.

[17] A. Arora, R. Krishnan, R. Telang, and Y. Yang, "An empirical analysis of software vendors' patch release behavior: impact of vulnerability disclosure," *Information Systems Research*, 2010.

[18] A. Nappa, R. Johnson, L. Bilge, J. Caballero, and T. Dumitras, "The attack of the clones: A study of the impact of shared code on vulnerability patching," in *IEEE Symposium on Security and Privacy (SP)*.

[19] L. Allodi and F. Massacci, "Attack potential in impact and complexity," in *International Conference on Availability, Reliability and Security*. ACM, 2017.

[20] S. Treetippayaruk and T. Senivongse, "Security vulnerability assessment for software version upgrade," in *IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, 2017, pp. 283–289.

[21] C. Xiao and A. Sarabi, "From patching delays to infection symptoms: Using risk profiles for an early discovery of vulnerabilities exploited in the wild," in *USENIX Security Symposium*, 2018, pp. 903–918.

[22] P. B. Lamichhane, L. Hong, and S. Shetty, "A quantitative risk analysis model and simulation of enterprise networks," in *IEEE Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, 2018, pp. 844–850.

[23] A. Singhal and X. Ou, "Security risk analysis of enterprise networks using probabilistic attack graphs," in *Network Security Metrics*. Springer, 2017, pp. 53–73.

[24] F. Yamaguchi, F. Lindner, and K. Rieck, "Vulnerability extrapolation: assisted discovery of vulnerabilities using machine learning," in *USENIX conference on Offensive technologies*, 2011, pp. 13–13.

[25] G. Grieco, G. L. Grinblat, L. Uzal, S. Rawat, J. Feist, and L. Mounier, "Toward large-scale vulnerability discovery using machine learning," in *ACM Conference on Data and Application Security and Privacy*.

[26] S. M. Ghaffarian and H. R. Shahriari, "Software vulnerability analysis and discovery using machine-learning and data-mining techniques: A survey," *ACM Computing Surveys*, vol. 50, no. 4, p. 56, 2017.

[27] J. A. Harer, L. Kim, R. L. Russell, O. Ozdemir, L. R. Kosta, A. Rangamani, L. H. Hamilton, G. I. Centeno, J. R. Key, P. M. Ellingwood, M. W. McConley, J. M. Opper, S. Chin, and T. Lazovich, "Automated software vulnerability detection with machine learning," 2018.

[28] R. Russell, L. Kim, L. Hamilton, T. Lazovich, J. Harer, O. Ozdemir, P. Ellingwood, and M. McConley, "Automated vulnerability detection in source code using deep representation learning," in *IEEE Int'l Conf. on Machine Learning and Applications (ICMLA)*, 2018, pp. 757–762.

[29] M. Zolanvari, M. A. Teixeira, L. Gupta, K. M. Khan, and R. Jain, "Machine learning based network vulnerability analysis of industrial internet of things," *IEEE Internet of Things Journal*, 2019.

[30] D. 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*, 2008.

[31] "Cvss," *"https://www.first.org/cvss/v2/guide"*.

[32] S.-H. Liao, "Expert system methodologies and applications—a decade review from 1995 to 2004," *Expert systems with applications*, vol. 28, no. 1, pp. 93–103, 2005.

[33] "Cpe," *https://nvd.nist.gov/products/cpe*.

[34] T. Lei, R. Barzilay, and T. Jaakkola, "Rationalizing neural predictions," *arXiv preprint arXiv:1606.04155*, 2016.