Building a Machine Learning Model for the SOC, by the Input from the SOC, and Analyzing it for the SOC

Awalin Sopan Matthew Berninger Murali Mulakaluri Raj Katakam

FireEye, Inc.

{awalin.sopan, matthew.berninger, murali.mulakulari, raj.katakam}@fireeye.com

ABSTRACT

This work demonstrates an ongoing effort to employ and explain machine learning model predictions for classifying alerts in Security Operations Centers (SOC). Our ultimate goal is to reduce analyst workload by automating the process of decision making for investigating alerts using the machine learning model in cases where we can completely trust the model. This way, SOC analysts will be able to focus their time and effort to investigate more complex cases of security alerts. To achieve this goal, we developed a system that shows the prediction for an alert and the prediction explanation to security analysts during their daily workflow of investigating individual security alerts. Another part of our system presents the aggregated model analytics to the managers and stakeholders to help them understand the model and decide, on when to trust the model and let the model make the final decision. Using our prediction explanation visualization, security analysts will be able to classify oncoming alerts more efficiently and gain insight into how a machine learning model generates predictions. Our model performance analysis dashboard helps decision makers analyze the model in signature level granularity and gain more insights about the model.

Keywords: Cyber security, Machine Learning, Information Visualization, Security Operations Center.

Index Terms: K.6.5 [Management of Computing and Information Systems]: Security and Protection; H.1.2 [Information Systems]: User/Machine Systems.

1 INTRODUCTION

In a security operations center or SOC, security analysts detect and triage time-sensitive security threats that require their full attention. One big challenge they face is the number of false positive alerts from various data sources. Analysts working in SOC have to investigate these benign alerts, low in impact but high in volume, which reduce the capacity of the SOC and erode analyst morale [1]. These alerts require some analysis, but not the creative and expensive mind of an analyst. Moreover, many detection rules or signatures are not very efficient and can cause the generation of many false positive alerts. Use of machine learning models to predict the maliciousness of such attacks, and thus potentially proactively identify false positives, can reduce analysts' workload. However, for such mission-critical tasks, analysts cannot solely depend on the machine learning systems, especially since there are always new types and sources of attacks. Even with a deployed, and accurate machine learning model, human experts are needed in the loop to make and explain the final decision.

For this project, we worked with security analysts to understand their workflow and mental model, then developed a machine learning model to aid their analysis. We built the model to classify alerts into two categories, "threat" or "false positive" (detected by the system as a threat but actually benign) along with a prediction score. We then built a system that provides this prediction to the human analysts along with an *explanation* of why it made that prediction. In this work, we demonstrate an ongoing effort to explain the alert classification by a machine learning model to the SOC analysts using a prediction explanation visualization. While a human in the loop approach in machine learning can help improve a model [2], most work has focused on interpreting and visualizing the model features for data scientists and engineers. Here we are focusing not only on the data scientists, but also on the analysts who triage alerts based on raw alert data as well as the model's prediction. Therefore, we developed a system that serves these purposes:

1. To provide analysts a prediction for the alerts from a machine learning model that uses features similar to the analyst's mental model
2. To enable the end-users, the analysts, to use the model's output with confidence
3. To enable the model developers and stake holders to validate and correct the model with a better understanding of its performance

For these objectives, we created a visualization of the model prediction to aid the analysts without overwhelming them. We have set up a system that generates metrics for the model and use an off-the shelf visualization tool and used its data visualization capability to transform information into insight. Our main contributions are the following:

1. The *machine leaning model* enables analysts and SOC managers to detect false positive alerts and prioritize alerts based on the threat prediction score
2. The *model prediction view* enables the analysts to:
   - Get an overall picture of the entire dataset though the visualization
   - Focus their attention on critical alerts
   - Add confidence for their decision, and perhaps question their logic if the model disagrees
3. The *model performance dashboard* enables the managers and data scientists to:
   - Get insights on where the models are doing better and where they need improvement.
   - Identify opportunities where the model can automate the triaging process and reduce analyst processing time
   - Decide which indicators of compromises are better for detecting genuine threats and which are generating a lot of false alerts.

2 RELATED WORK

With recent advances in machine learning paradigms, we have noticed a great interest in interpretation of machine learning models. Much recent research focuses on model interpretation [3, 4] while also creating the best performing model, visualizing the

978-1-5386-8194-7/18/$31.00 @2018 IEEE

training data distribution, and illustrating the behavior of the model. A visual analytics approach is also often used to compare two models being used for the same dataset, in order to decide which models to use in a particular case for properly predicting labels [5]. In the case of neural networks, researchers have built applications to visualize the network itself [6]. Our work relates more to the nature of interpretation of instance level, traditional model decision making [7, 8] where one can understand which features of a particular instance (say, an alert) are used by a model to make a decision. Prior work on this aspect used highly interactive and complex visual interfaces to analyze the prediction [9], but security analysts often do not have enough time to interact with such complex visualization. We needed to create a simplified visualization that can both provide confidence without overwhelming the security analysts. It is also very important to understand their workflow and how they perceive a security threat [10]. Inspired by DARPA's Explainable AI (XAI), Vigano' et al. [11] defines several possible pathways to Explainable Security noting that explanation can help increase confidence, trust, transparency, usability, verifiability and accountability. Use of machine learning for malware analysis and intrusion detection helps identify potential security threats [12]. If we present the model's prediction along with appropriate context, the analysts can even provide feedback to the model re-training process by selecting wrongly classified alerts [13]. However, if only presented with a model's prediction without any explanation, analysts will lose confidence when the model is wrong. Moreover, if the model uses abstract features from the data that the analysts are not familiar with, the interpretation will make no sense to them, and – even if correct - trust between analyst and model will suffer. To combat these challenges, our work focuses on using features that the analysts look at while triaging an alert and designing a simplified user interface that they can quickly glance over. We recognize that analyzing machine learning models is not their main task and we cannot deviate them from their regular workflow with complex visualization. Businesses are employing machine learning applications more often than ever, and to the upper management, the impact of such applications comes with the question of cost reduction and efficiency [14]. They are more familiar with commercial business intelligence tools such as Tableau [15]. However, there is a new emerging market where commercially available machine learning suites not only provide faster model development techniques, they also provide understanding of the model itself, like H2O.ai [16]. When creating such applications explaining machine learning models and their performance, we need to keep in mind who are the users and what are their goals.

3 System Description

We have worked closely with the SOC analysts of a commercial cyber security organization. The SOC analysts use a web-based analysis platform to investigate cyber security alerts from various customers of their organization. These alerts are generated from some network or endpoint devices, triggered by some signature or indicator of compromise (IOC) written by cyber security experts. Analysts can view the raw data of the alert and pivot around various features, as well as request more information regarding the alert before identifying if a threat is genuine or not. After they make their decision about the alert, they tag each alert either as a 'Threat' or a 'False Positive' one. To build the machine learning model, we have used these tagged alerts as training data for our model. We convert each of these alerts into a vector of features, and we use the analyst-given tags as the ground truths for training the model. The workflow is depicted in Figure 1. A signature-based detection

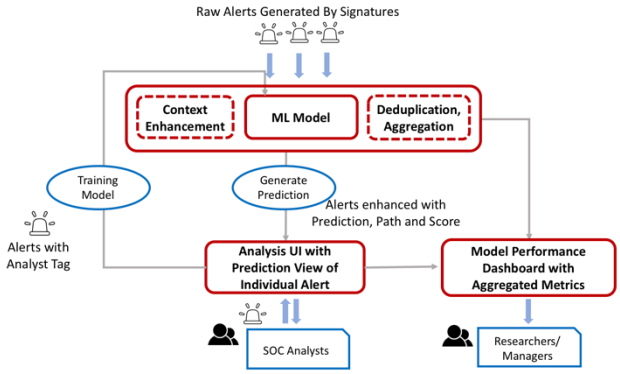

Figure 1: System Workflow

system detects alerts from various sources and sends it to the analysis pipeline. There the alerts are first enhanced with more context, aggregated and deduplicated (shown with dotted line since these phases are out of the scope of this paper), and are passed through a machine learning model that classifies the alerts as 'threat' or 'false positive'. Then these alerts are presented to the SOC analysts along with the prediction label. The analysts tag these alerts as 'Threat' or 'False Positive'. These analyst-given Tags are the final decision regarding the alerts; these decisions are used to triage the alerts and also fed back into the system to retrain the model. Both Analyst Tag and Prediction Label along with raw alerts data are fed into another dashboard that generates a live overview of the model's performance and is presented to the stakeholders.

3.1 The Model

Before developing the system, we interviewed five of the most experienced analysts in the company with expertise on areas such as analyzing network traffic, infected host, and sandbox analysis of malware binaries. We conducted total of 6 individual interview sessions, each ranging from 1 to 2 hours. In those sessions each of the analysts showed how they search for alerts, look at the details of an alert, use different tools available at their disposal to get more information on more difficult alert decisions, and then finally decide whether the alerts are a legitimate threat, or a false alert triggered by a signature. The interview sessions covered alerts generated by email content, network traffic, potential malicious processes invoked in hosts, and potential malicious files written in hosts. We recorded what aspect and information of an alert help them make a decision. Then we codified those aspects into 'features' for our prediction model so that the model can represent the analysts' decision-making criteria. The central idea behind building features for our alert classification model was to find a way to represent and record all the aspects that an analyst might consider when making a decision. For example, consider a process execution event on a computer. An alert on a potentially malicious process execution may contain the following fields: Process Path, Process MD5, Parent Process, and Process Command Arguments. While this may initially seem like a limited feature space, there is a lot of useful information that one can extract from these fields. Beginning with the process path of, say, "C:\windows\temp\m.exe", an analyst can immediately see some features:

- The process resides in a temporary folder: C:\windows\temp\
- The process is two directories deep in the file system
- The process executable name is one character long

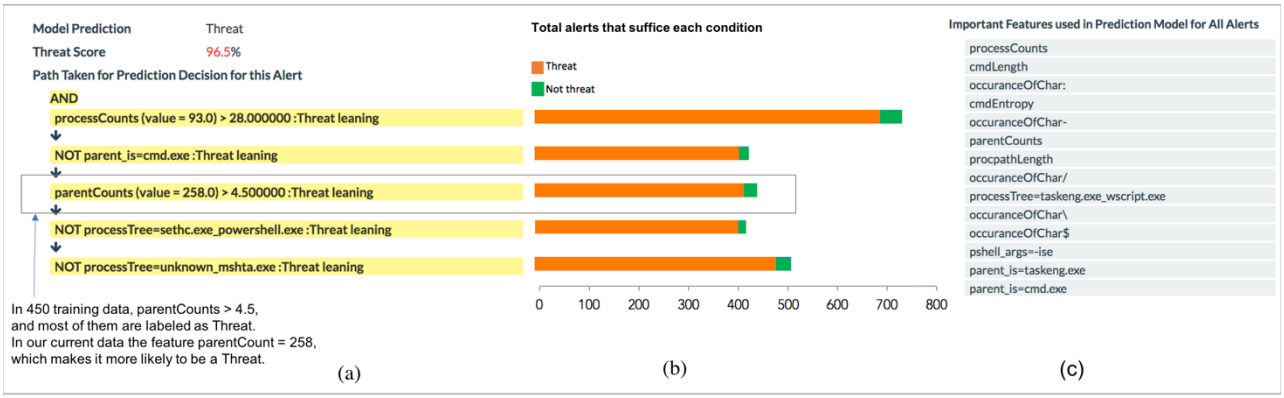

Figure 2: The Prediction View UI Component: The alert is classified as a Threat, with 96.5% probability by the model. The conditions fulfilled by the model to come to the decisions are highlighted in yellow; the first condition is 'processCount' = 93 which is greater than 28, and in our model a processCount value greater than 28 is considered as a highly likely threat.

- The process has an .exe extension
- The process is not a "common" process name

While these may seem simple, over a vast amount of data and examples, extracting these bits of information will help the model to differentiate between events. Even the most basic aspects of an artifact must be captured in order to "teach" the model to view processes the way an analyst does. The features are then encoded to this discrete representation:

Temp folder	Depth	Name Length	Extension	Common Process Name
TRUE	2	1	exe	FALSE

Each alert A is converted into a feature vector A'=[f_1, f_2, f_3, …, f_n] and passed to the model. Since we prioritized on prediction interpretability, we decided to use models that are more interpretable and easily explainable. Therefore, for the underlying model, we implemented a Random Forest Classifier consisting of more than 200 decision trees using Scikit-Learn platform. The model is thus agnostic of the alert generation process and uses the analyst-driven features that are easily explainable to the analysts. We have trained our model with alerts observed over the previous six months and used the analyst-given tag as their training label. During training we use 90% of the data as the training set and 10% as the test set. In the training dataset we used more than thirty thousand alerts labelled as Threat and around ten thousand labelled as False positive; to overcome this imbalance, we assigned double weight to all the False Positive training alert instances. On our last deployed version of the model, the model's accuracy was little over 98.5% in the test data.

3.2 The Prediction View

We developed the user interface component based on analyst feedback as well. Our user interface by default only shows the score given by the model and the decision. The analyst can click a button to request to see the decision path, and it is shown on demand. This design decision was made so that the interpretation view will not crowd the screen and the analysts will be able to see the model interpretation only when they want to. Rather than creating a whole new application, we have made a component that is integrated with their analysis app and their workflow. The same application that shows then the raw alert data and the related information, is now enriched with the model prediction and interpretation.

Before we made this prediction label available to the analysts, we wanted to make it reliable and interpretable, so they can understand when to trust the model. Our current work builds on that goal: we created a user interface (UI) component that shows the analysts what the underlying machine learning model thinks of the alert: whether it is a threat or not, and 'Why'. This Prediction View has the following components (Figure 2, 3):

1. The classification label produced by the model and the threat score (the confidence score used to gauge if alerts are a threat). If

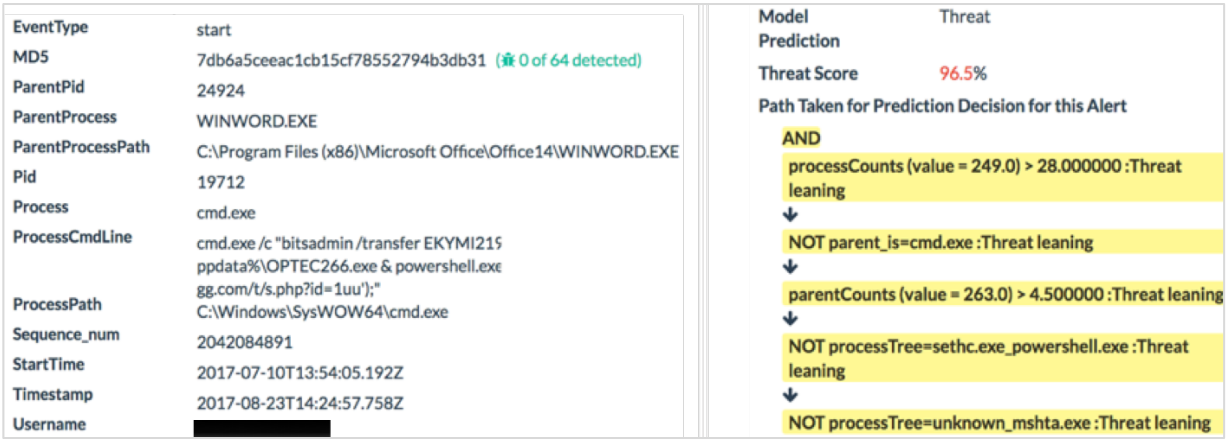

Figure 3: Left: Alert's raw data presented to a SOC analyst. Right: Prediction View for the same alert.

the score is above 50% for an alert, the model classifies that alert as a 'threat'

2. The decision path to reach that classification: what features of the current alert are used by the model and how they play in the situation

3. The main 'features' of the alerts for the overall training set used by the model

We followed a visual information hierarchy to present the model's decision to the analyst: at the very top of the view, we show the model's prediction decision and then the prediction score as 'Threat Score'. If the score is below 80% we render it in grey text, above 80% but below 95% in orange, and beyond 95% in red (highly likely to be an actual threat). This is the most important piece of information for the analyst and using this color coding help draw their attention. The Decision Path is hidden by default, and the analysts can see it on demand by clicking a 'Show Decision Path' button (not shown in the screenshot). The raw data from the alert itself is also presented in the same screen next to the Prediction View, so the analyst can compare the features in the Prediction View and the alert details. We decided to display the prediction in the context of the alert itself, so the analysts can easily view the features used by the model, compare that with the alert itself and verify that looking at its decision path (the conditions highlighted in yellow) (Figure 2, leftmost panel, (a)).

While the deployed model uses an ensemble of decision trees, we do not want to overwhelm the analysts with a visualization of all the trees as in [9]. This type of in depth tree-by-tree analysis is useful for debugging a model, but SOC-analysts do not have time for such exploration. In addition, we realized this view should be model agnostic; if we decide to use a different type of model in future, the visualization should remain the same, and the analysts will not have to retrain themselves with a new UI. Therefore, when we show the alert decision to the UI, we run the alert again through a simple representative decision tree to get the decision path this tree has taken to reach a decision node. We have noticed a long tail of Feature Importance distributions where most of the features have a very low feature importance score. We extracted the most important features (features with a Feature Importance value of at least 0.005) from the random forest model, then created a representative Decision Tree classifier (with one decision tree) using those features. Finally, in the UI, we show the decision path taken by this classifier. This decision tree is a simple representative of all the trees the model used. The branches of the decision tree that lean towards deciding the alert as a threat are shown in yellow and the ones that are false positive leaning, are shown in green (Figure 4). This way our threat score and classification decision come from a more balanced random forest model with 200+ decision trees and the simplified decision path comes from a single decision tree. An analyst can also compare this decision with the overall data set (the visualization of the data distribution for each matched condition) (Figure 2, middle panel). The horizontal bar chart shows how many of the training alerts fell under the same condition; we break them down by their true label, and color code them by label. In the given example, we have seen 450 alerts that have 'parent Counts' >4.5, and around 400 of them were Threat (orange) and 50 of them were not (Blue), since our current alert has 'parent Counts' way above that, there is a significant chance that this is a threat.

The right-most panel in Figure 2 shows which features are deemed the most important ones by the model; not just for this instance, but for all training data. The model considers all the features but not all features are equally important. If an alert has a blacklisted domain, or a malicious MD5-hash is associated with it, then the model can be more confident about the alert's maliciousness. For decision tree-based models, feature importance [17] can be measured by calculating their Information Gain, or their Gini Impurity. We show the ranking of these features to give our analysts a simple idea of which features of the alerts are more distinguishing than others. Since an analyst only looks at one alert at a time, this overview presents itself with an opportunity to gain knowledge about the entire dataset that an analyst may not have previously considered. The threat score not only helps the analysts make decisions quickly, it also helps prioritize and automate their decision process. If an alert is classified as a threat by our model with high confidence, an analyst can, with confidence, quickly make a decision and move on to focus their attention to a more critical, or perhaps more ambiguous alerts.

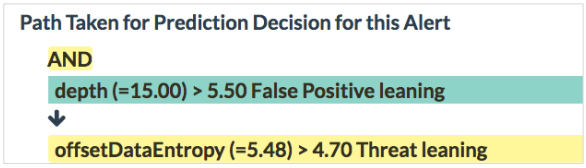

Figure 4: Segment of a decision path showing one branch with False Positive leaning, and the next branch to be Threat leaning.

Nonetheless, machine learning models can make mistakes too, due to insufficient training data, complexity of alerts, new types of attack vectors, etc. Expert analysts can suggest improvements to a model by providing their feedback. Our UI provides a mechanism for the analysts to tag the alerts with their final decision, and then it feeds this tag back to the model. If the analysts find problem with the decision path, they can input their comments to the system regarding the decision.

To summarize, at the time of investigating an alert, the analysts can see the raw data of the alert along with alert related context (the source, client organization, time of the alert, etc.), the prediction from the model, an approximation of the decision path, and a simplified, interpretable view of the overall feature importance.

3.3 The Model Performance Dashboard

After the trained model is deployed in production, it started to predict labels for new alerts. These new alerts are shown to the analysts along with prediction score as described in the previous sub-section. Now we need to see how the model performs on the new alerts which will validate the model's efficacy. To get a deeper understanding of the model, and analyze where the model is performing well enough to trust it completely, we created a model analysis dashboard. The paper shows three months of these new alerts in the dashboard. We store both the model's prediction and the analysts' final decision of the alert, that way we possess both Model's Prediction Label as well as the Analyst's Tag. Using this dataset, we created a visualization dashboard for the managers and stake holders who want to see the overview of the model efficacy and take the machine learning (ML) model application further. For this dashboard we use Tableau server. The simplest metric to gauge the model's performance is to check how often the model is correct, i.e., how often the analysts agree with the model's prediction. We used the model Accuracy measure for that, and labelled it as ML Success in the dashboard for better compensability to managers and business-decision makers who are not necessarily experts on Machine Learning paradigms:

ML Success = [(Analyst Tag == False Positive AND Model Label== false positive) OR [(Analyst Tag == Treat AND Model Label== threat)]

For the last three months of data, we observed that ML Success or Accuracy is 90% on this production data. But is the model more successful classifying threats or is it failing to detect them? To answer this question, we break it down by Analyst-given Tag (Figure 5 Top). This visualization shows that the model is usually correct in classifying malicious alerts as threat; only in 13% cases it wrongly classified a genuine threat as a false positive one. Conversely, it failed to classify a false positive in 69% of cases. While the success rate itself gives a very optimistic picture, we want to examine how confidently the model is making these decisions. When it is correct, how confident it is about its correctness? When it fails, is it still very confident? If it classifies a genuine 'threat' with 90% confidence that it is a 'false positive', we need to reconsider using such a model. To do such analysis, we decided to break down the ML Success by Prediction Score buckets (Figure 5, Center). The prediction score buckets are: a) 95 and up, b) 80 to 95, c) 65 to 80 and d) 50 to 65 (for binary classifier, 50 is the lowest possible score). This shows the higher prediction buckets are more likely to be correct. When the model is not confident enough it is more likely to misclassify an alert. This is a good sign that even when the model is wrong, it is not wrong with high confidence. For a more in-depth analysis, we combined the two approaches: we obtained a visualization where the Analysts Tags are broken down into buckets of model Prediction Score (Figure 5, Bottom). The higher score buckets show better results for both Threat and False Positive tags. For the highest score bucket, 95 and up, it shows 97% accuracy for False Positive Tags and 100% for Threats, indicating that when the model was 95% or more confident that an alert is a Threat, it was correct (Figure 5, Bottom). The sudden drop in accuracy in False Positive 65 to 80 bucket (Figure5, bottom, (c)) caught our attention, and after filtering these alerts we found out most of these misclassified alerts are coming from a bad signature that caused a surge of alerts. Our analysts removed that signature from detection engine and decided to discard those alerts from the next iteration of the model training.

No machine learning model analysis is complete without analyzing the confusion matrix. We followed a combination approach using a confusion matrix and the Prediction Score buckets (as shown in Table 1). Rather than showing a simple confusion matrix, we created a bucketed confusion matrix (Table 1). This table view is converted into stacked bar charts in Figure 6. If the score is 80% with a prediction label of Threat, then the model is highly confident that the alert is malicious. While this table (Table 1) itself gives us some idea on how the model is performing, we can get a better picture with a bar chart showing how the model is doing overall (Figure 6). This chart shows the model Prediction Label as columns. The percentage of alerts (top) and total number of alerts (bottom) are color coded by Analyst Tag (ground truth). In an ideal case (model label and Analyst Tags are the same), all the bars under the Prediction Label 'false positive' will be green, and all under 'threat' will be orange, indicating that the model is correctly classifying both the labels. But in reality, we see that, most of the alerts that are correctly predicted as a threat (orange, right column), fall in the higher confidence bucket. On the other hand, while the model correctly predicted many alerts as false positive (green in the left column), it is still incorrectly predicting a lot of alerts as false positive (orange bars in false positive column). The '80 to 95' bucket in the 'false positive' Prediction Label column is especially concerning, since the model is wrongly classifying these false positive alerts (the orange portion of the bar) as threat with such high confidence.

		Analyst Tag (=ground truth)			
		% of Alerts		Total Alerts	
Prediction Score	**Prediction Label**	False Positive	Threat	False Positive	Threat
<=65	falsepositive	16.57	83.43	116	584
65 to 80		26.14	75.00	23	66
80 to 95		13.70	86.58	97	613
>95		100.00	0.00	335	0
<=65	threat	12.75	87.35	128	877
65 to 80		35.94	64.17	649	1159
80 to 95		1.29	98.71	74	5677
>95		0.19	99.85	10	5280

Table 1: Bucketed confusion matrix: Showing both % and counts, broken down by Prediction Score, grouped by Analyst Tag.

To understand the actual scale of the issue, we also show the total number of alerts below the percentage chart and notice that, the actual number of alerts that are tagged as false positive by

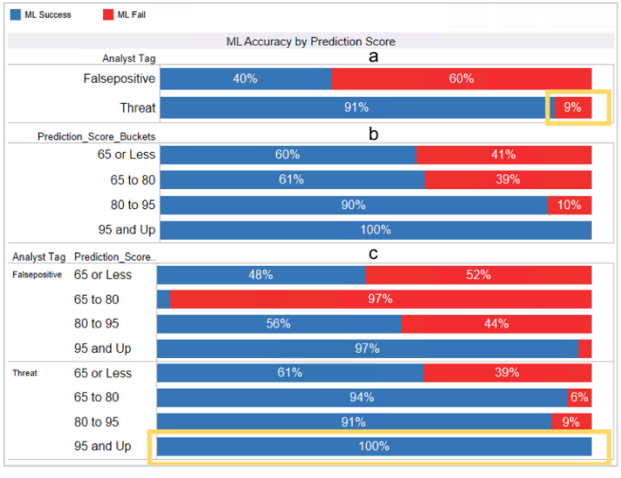

Figure 5: Top (a): Model Accuracy by Analyst Tag. Center (b): Model Accuracy by Prediction Score Bucket. Bottom (c): Model Accuracy by Analyst Tag broken down by Prediction Score Bucket.

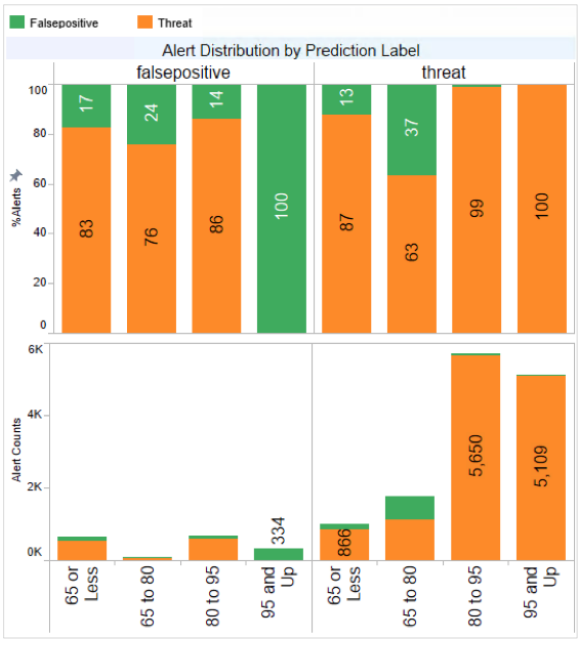

Figure 6: Distribution of Prediction Label over Prediction Score buckets; color-coded by Analyst Tag (Ground Truth). Top: Percentage of False Positive and Threat under each bucket, Bottom: Total count of alerts under each bucket. In ideal case, all bars under 'falsepositive' should be green, and all bars under 'threat' should be orange.

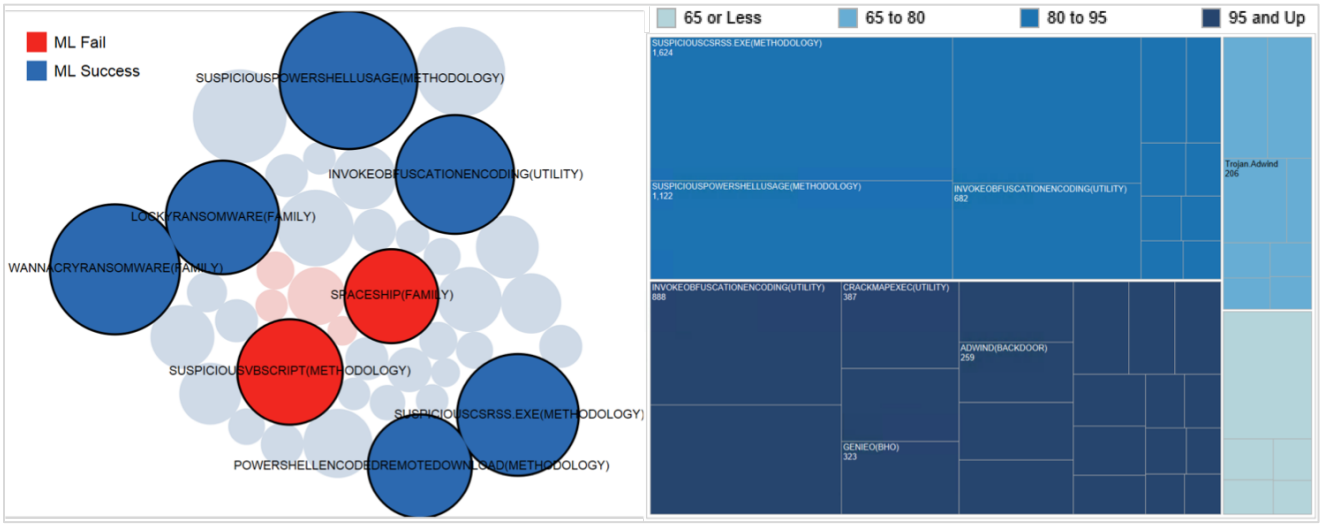

Figure 7: Left: Circle pack visualization showing alerts by signatures. Each circle represents alerts from a particular signature. Circles are sized by the total alerts of that signature and color coded by the ML Model success of ML failure. Right: A Treemap visualization showing only alerts that are correctly labeled by the model, grouped by signatures. Color coded by prediction Score range, sized by total number of alerts in that signature group. It shows which signatures are more common and how the model is performing to classify alerts triggered by those signatures.

the analysts are actually not that many. This was also the case in our training data, so when we trained the model we had an imbalance of false positive (benign) vs threat (malicious) data. That made us focus on improving the next iteration of the model carefully and train it with more false positive examples. This bucketed confusion matrix view gives us an overall idea, and in this case, we see that the model is doing well identifying threats but has room for improvement when it comes to identifying non-threats.

However, this information is not enough to automate the decision-making process such that we can show only the critical alerts to the analysts and use the model's prediction for high confidence labels. We need more benign training data to detect the false positives with better precision. We have already performed granular analysis on the outputs of the model - the level of prediction label and prediction score. It is also possible that the input characteristics of the alerts themselves are a key-player for various levels of prediction accuracy. Since the signatures are the reason why these alerts were triggered at the first place, we now want to see how the model performance varies by different signatures. Alerts generated by different signatures have different characteristics, so this seems to be a natural grouping. If the signatures are very efficient/accurate, then we will seldom have false positive alerts triggered by that signature. The analysts wanted to know from the system developers which signatures are successfully predicting alerts and which ones are bad signatures creating lots of false positives. Motivated by their query, we break down the data by signatures. The circle-packed bubble chart visualizes alerts grouped by signatures and color coded by ML Success and Failure (Figure 7, Left). The size indicates the total number of alerts triggered by that signature. For demonstration purposes, we have highlighted a few of them (unselected ones are transparent in the background). We see that most of the signatures are blue, indicating that the model is making successful predictions on alerts triggered by those signatures. If we can identify the signatures where the model is consistently performing accurately with a high score, we can consider automating the decision-making process for those signatures. For this purpose, we select all the alerts that are correctly classified by the mode land show a Treemap visualization (Figure 7, Right) of the model accuracy scores for those alerts, grouped by signatures. Rectangles are sized by alerts from each signature, and color coded by the average Prediction Score of the prediction model for that signature (darker shade of blue means higher Prediction Score). The average Prediction Score bucket and the alert counts are also shown inside the Treemap nodes. It shows, for some signatures, the model's accuracy is very close to 100%, whereas in some cases (rectangles with lighter shade of blue) the model is not performing well enough to be trusted completely. Larger rectangles with a darker shade of blue indicates that not only we are getting a lot of alerts from this signature, our model is also performing very well in terms of classifying alerts from those signatures. In such cases, we can trust the model to the point of automating the triaging of those alerts to reduce the workload of the analysts.

For a detailed picture of the model performance by signatures, we generated a table view with bucketed model prediction score. Each row (Figure 8, Left) represents a signature and the columns show the total number of alerts predicted by the model in particular prediction score bucket; the size of the rectangle represents the alerts count and the color represents model's success/failure. We notice several signatures have both red and blue rectangles (Figure 8, Right), i.e., these signatures are triggering alerts that are confusing our model, some of them are predicted as threat and some others as false positive. We will need to focus on alerts from those signatures in the next iteration of the model retraining. This detailed and information-rich view gives a quick overview of the entire signature set. helps us identify signatures where:

1. The model is always correct: completely trust the model
2. The model is often wrong: cannot trust the model at all
3. The model has both correct and wrong labels: some tweaks may improve the model
4. The model and the analysts both classify the alerts from these signatures as False Positives

For the last category, we get the signatures for alerts that the model correctly labels as False Positive. These are bad signatures and should be removed from the alert detection process. By identifying the signatures from these different categories,, we get one more step closer to automating the final decision using the model prediction and remove bad signatures causing more false

Figure 8: Left: Table showing Prediction Scores and Model Accuracy for all Signatures, color coded by ML Success and Failure, Rectangles sized by Alert count. Right: Filtered signatures where the model is predicting all or some alerts incorrectly.

positive alerts. If the model is particularly accurate on a specific type of alert, then the SOC can write a rule and use a rule engine to check if the alert falls under such rule and automate the alert processing. For example, a rule can be, "*If the alert possesses XX signature, AND the model confidence is above 99, automatically use the model's decision as the final decision.*" - and this alert will not be shown to the analysts. If a signature is generating a lot of false positive alerts, we can remove those from alert generation.

4 Results and Feedback

In the first iteration, the Prediction UI showed model's prediction score as 'Prediction Score' and it confused analysts who are not familiar with machine learning terminology. We then changed the UI label to 'Threat Score' since this is what the model is predicting. Rather than showing "70% chance to be a false positive", we changed it to "30% chance to be a threat". Therefore, when the model's binary classifier predicts a 70% confidence with False Positive, the interface shows Threat Score 30%.

After the model and Predict View deployment, we conducted feedback sessions with the analysts explaining the decision tree model and the decision path paradigm. One analyst emailed the first author to learn more about the model and how it makes decisions. After the model deployment, the authors have reached out to the analysts to understand how and if the prediction score is being used. One analyst mentioned, '*We are actually looking at the score and that has been very useful*'. Another one mentioned that the decision path itself is extremely helpful. Several analysts wanted to use the prediction score to write an automation rule that will automate their investigation decision for future alerts with a score above a given threshold. After that feedback, we have implemented that feature in the UI so that an analyst can create a rule that will match all alerts that are predicted.

When the analysts disagree with the model's decision they can enter a comment in the system explaining why they disagree; the data scientists can use that feedback to improve the model for future alerts and determine the outliers. This way the analysts can provide insight regarding the model without getting into the mathematical details. For example, one analyst mentioned, *'I saw the following alert in which the predicted label is False Positive with a 93.1% confidence. However, the alerted file is malicious and detected by a number of AV vendors on VT.'* After receiving the feedback on AV vendors, we realized that the feature set are not sufficient for all alerts and decided to incorporate Virus Total [18] results (how many vendors detected a given MD5 hash as malicious) of detected malware as a feature for the model. Another feedback was, '*Check out this alert: Predicted at 60% probability to be a false positive but definitely looks bad.*' For this case, we figured out that it was a new type of alert from a new signature and we did not have similar alerts in the training data, resulting in an inaccurate prediction.

Prior to creating such automation rules, the analysts need to know if they can trust the model for such misclassifications. We have seen that new types of alerts can cause the model to fail. Hence the analyst wanted to see a distribution of prior predictions for alerts of the same type or from the same signature. As a follow up, we designed a Confusion Matrix of the model for alerts of the same signature as the one they are investigating (Figure 9). This view shows how many alerts from that signature the model has seen and how it made predictions on them. This view will be presented to the analysts in the same Prediction View UI component in the near future, so they can decide whether to trust the model for that particular signature.

For the Model Performance Dashboard, we leveraged an off the shelf visualization tool (Tableau) since the interaction is familiar to the decision makers and is easily sharable. We have used familiar charts and tools for the purpose of analyzing and explaining machine learning model performance in finer granularity. SOC analysts are burdened with investigating alerts from hundreds of signatures, and not all of these signatures are properly able to detect genuine threats. They expressed to the authors that reviewing alerts triggered by those signatures are just time. Using the model dashboard, we were able to identify signatures that are consistently generating false positive alerts, and we decided to remove these signatures from generating alerts.

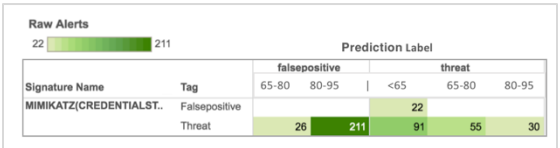

Figure 9: Prototype design of a Confusion Matrix for alerts from signature MIMIKATZ, color coded by alerts count, grouped by prediction score.

5 Conclusion and Future Work

In this work, we explore the human-computer dynamics of using machine learning models and the drive to gain more from the model. We developed a machine learning model trained by real alerts with labels given by SOC analysts working in their professional environment and implemented a feedback loop from analysts to the model to improve it. We are able to identify useful vs wasteful signatures. We have not done a full usability study on how the analysts are taking full advantage of the visualization; however, we have received positive feedback and improvement suggestions for future iterations. We acknowledge that different analysts have different investigation-approaches and may not look at the same alert features. We need to incorporate these different workflows while building the next iteration of the interpretable model. We are currently conducting an online survey among the SOC analysts to get a more structured feedback on the Prediction View. The survey questions are:

1. How often do you look at the Prediction View?
2. Is the decision path view helpful to you?
3. What more would you like to see out of this?
4. What do you use this Prediction View for?

Initially five senior analysts took the survey; three of them reported they use the prediction view more than half the time, and two use it almost always. Four of them said the decision path is helpful. The one analyst who said the path is not helpful, asked to see more context of the alert in their answer of question 3. One analyst who uses this view almost every time during their investigation, wanted to see more statistics around the prediction and training data, especially to understand why their decision differs from the model. After we get the survey results from more analysts, we can get a better understanding of how they use the Prediction View and how we can better serve their investigation process using machine learning model's. The interpretation view helped the analysts and the data science team altogether by converting the model's decision into an interpretable one. We received their feedback which included suggestions such as using more features, showing more details of the prediction, and the ability in the UI to automate the investigation process based on the model threat score. The in-depth analysis of the model using the model performance dashboard helped identify the signatures that are producing more accurate predictions. For these signatures, we aim to automate the investigation decision.

SOC analysts are often inundated with high volume of low priority alerts. This automation process will expedite alert analysis and enable SOC analysts to dedicate more time towards more difficult problems. This will increase analysts' efficiency and reduce their workload. However, we do not want to falsely classify an alert to be a false positive when it can be a malicious attack in reality. Hence, we need to be careful before fully automating such system. We are also working on to determine approximately how much time we can save by using such automation.

A machine learning model may be good at detecting threats [19] but may fail to classify the benign ones, or vice versa. For different types of alert different threshold of confidence might be more applicable. Providing more context will help the analyst understand when to appropriately trust the model. We do not want to overwhelm our SOC analysts with visualizations like a Receiver Operator Curve [20], or metrics like F1 score or Matthews correlation coefficient; by using familiar terminologies and visualizations, we can democratize the output of machine learning models for the analysts and business-decision makers. However, we are working on developing new visualizations to give them more context. After gaining more familiarity with the machine learning models, advanced visualizations will be more meaningful to the analysts.

Acknowledgements

We cordially thank M. Cantos, N. Carr, P. Smith, D. Lindquist, K. Reichert, A. McCabe, C. Brooks, A. Vance, and all the analysts.

References

[1] Michael J Assante and David H Tobey. Enhancing the cybersecurity workforce. IT professional, 13(1):12–15, 2011.

[2] Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Why Should I Trust You?: Explaining the Predictions of Any Classifier. SIGKDD '16). ACM, NY, USA, 1135-1144.

[3] Naveed Akhtar, Ajmal Mian. Threat of Adversarial Attacks on Deep Learning in Computer Vision: A Survey, *Access IEEE*, vol. 6, pp. 14410-14430, 2018, ISSN 2169-3536.

[4] Gary K. L. Tam, Vivek Kothari, and Min Chen. An Analysis of Machine- and Human-Analytics in Classification. IEEE TVCG 23, 1 (January 2017), 71-80.

[5] Hossam Sharara, Awalin Sopan, Lise Getoor, Lisa Singh. G-PARE: A Visual Analytic Tool for Comparative Analysis. IEEE VIS 2011.

[6] Kanit Wongsuphasawat, Daniel Smilkov, James Wexler, Jimbo Wilson, Dandelion Mane, Doug Fritz, Dilip Krishnan, Fernanda B. Vie´gas, and Martin Wattenberg, Visualizing dataflow graphs of deep learning models in tensorflow, *TVCG*, vol. 24, no. 1, pp. 1–12, 2018.

[7] Josua Krause, Adam Perer, and Kenney Ng. Interacting with Predictions: Visual Inspection of Black-box Machine Learning Models. CHI '16. ACM, New York, NY, USA, 5686-5697.

[8] Josua Krause, Aritra Dasgupta, Jordan Swartz, Yindalon Aphinyanaphongs, and Enrico Bertini. 2017. A Workflow for Visual Diagnostics of Binary Classifiers using Instance-Level Explanations. VAST 2017.

[9] Marco Angelini, Leonardo Aniello, Simone Lenti, Giuseppe Santucci, and Daniele Ucci. The Goods, the Bads and the Uglies: Supporting Decisions in Malware Detection through Visual Analytics. VizSec 2017.

[10] Andrew M'manga, Shamal Faily, John McAlaney, & Christopher Williams. Folk Risk Analysis: Factors Influencing Security Analysts' Interpretation of Risk. SOUPS 2017.

[11] Luca Vigano', Daniele Magazzeni. Explainable Security. Workshop on Explainable Artificial Intelligence, IJCAI-ECAI, 2018.

[12] Anna L. Buczak and Erhan Guven. A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection. IEEE Communications Surveys & Tutorials, vol. 18, no. 2, pp. 1153-1176, Second quarter 2016.

[13] Bram C.M. Cappers and Jarke J. van Wijk. Understanding the Context of Network Traffic Alerts. VizSec 2016.

[14] Ajay Agrawal, Joshua S. Gans Avi and Goldfarb. Prediction Machines: The Simple Economics of Artificial Intelligence. Harvard Business Review Press, 2018.

[15] Tableau [Software]. Retrieved from www.tableau.com. [Accessed July 2018]

[16] H2O.ai [Software]. Retrieved from h2o.ai. [Accesses July 2018].

[17] Gilles Louppe, Louis Wehenkel, Antonio Sutera, and Pierre Geurts. Understanding variable importances in forests of randomized trees. NIPS'13. Curran Associates Inc., USA, 431-439.

[18] VirusTotal.com [Accessed 24 May 2018].

[19] LeDoux C., Lakhotia A. (2015) Malware and Machine Learning. In: Yager R., Reformat M., Alajlan N. (eds) Intelligent Methods for Cyber Warfare. Studies in Computational Intelligence, vol 563. Springer, Cham.

[20] Tom Fawcett. An introduction to ROC analysis, Pattern Recognition Letters, Volume 27, Issue 8, 2006, Pages 861-874, ISSN 0167-8655.