MEASURING CONCEPT DRIFT IN MALWARE AND NETWORK INTRUSION DETECTION MODELS

BY

ZHENNING ZHANG

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois Urbana-Champaign, 2024

Urbana, Illinois

Adviser:

Associate Professor Gang Wang

ABSTRACT

This thesis delves into the phenomenon of concept drift, a critical issue in the field of machine learning where the statistical properties of the target variable, which the model is trying to predict, change over time. This work is particularly focused on understanding the reason and impact of concept drift in cybersecurity contexts through measurement and modeling approaches, using both Portable Executable (PE) files in Windows and Android malware datasets, as well as network data from a real-world Security Operations Center (SOC) facility.

The research begins with a comprehensive introduction to the concept drift, laying out its definitions and taxonomies, specifically highlighting feature drift and data drift. It then proceeds to explore these types of drifts using a PE/Android dataset, analyzing how feature and data drifts manifest in these domains.

Subsequently, the thesis introduces a novel model designed to detect data drift in network traffic. The model employs a unique approach to generate cross-host features and utilizes a Support Vector Machine (SVM) for the detection of data drift. Meanwhile, we also perform measurements to understand the network attacks.

Through rigorous analysis and modeling, the research presents a concrete step to the understanding of the reason and impact of concept drift in cybersecurity, presenting a novel approach to network intrusion detection that can be beneficial for future research and practical applications in the field. The findings not only enhance the academic understanding of concept drift but also offer practical solutions to detect and adapt to it in dynamic environments, thereby improving the robustness and reliability of machine learning models in security-sensitive applications.

ACKNOWLEDGMENTS

At the culmination of this journey, I bid farewell to the eighteen years of my academic voyage—a chapter marked by countless challenges surmounted and victories cherished. To every obstacle that stands in my path, I am grateful for the resilience it instilled in me.

To my parents, whose unwavering support and boundless encouragement have been the cornerstone of my success, I owe a debt of gratitude beyond measure. Their belief in me, even when I faltered, has been my guiding light.

To my girlfriend, whose love and understanding have been a constant source of strength, thank you for standing by me through the highs and lows of this endeavor. Your presence has made every hurdle seem surmountable.

To all my friends, who have shared in both the joyous moments and the burdensome trials, your camaraderie has made this journey not only bearable but also immensely fulfilling. Your encouragement and camaraderie have been a source of motivation beyond words.

Finally, I extend my heartfelt thanks to my advisor, Professor Gang Wang, whose guidance, expertise, and unwavering support have been invaluable throughout this endeavor. I am deeply grateful for the wisdom and encouragement you have shared, which have undoubtedly shaped the trajectory of my academic pursuits. You are the best professor I have ever met.

To all my coworkers, whose collaboration and camaraderie have enriched my learning experience, thank you for your contributions to this work.

This thesis stands as a testament to the collective support, encouragement, and mentorship I have been fortunate enough to receive. Each of you has played an integral role in shaping both my academic journey and my personal growth, and for that, I am profoundly grateful.

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

In recent years, machine learning (ML) models have increasingly become the backbone of various cybersecurity measures, including the detection of malicious software (malware) and network attacks. Researchers and practitioners alike have endeavored to construct sophisticated malware detectors or network intrusion detectors leveraging ML techniques, as evidenced by significant works in the field [1, 2, 3, 4, 5]. Despite the advancements, a persistent challenge these detectors face upon deployment is the phenomenon known as concept drift. Concept drift refers to the scenario where the decision boundaries of ML models shift due to changes in underlying data distributions. This shift is particularly problematic for ML-based malware detectors because these models are predicated on the assumption that the data encountered in operational environments will closely mirror the data on which they were trained. However, in real-world applications, the distribution of testing samples may evolve organically or as a result of deliberate adversarial actions, leading to a degradation in model performance over time [6]. Cyber attackers continuously develop new strategies and modify existing malware to evade detection, leading to changes in the underlying data distribution that ML models rely on.

Recognizing the critical impact of concept drift, researchers have explored various strategies aimed at detecting and mitigating its effects. Some studies focus on identifying instances of drift in malware samples [7, 8, 9, 10], while others examine the aging of models over time and propose methodologies for their rejuvenation [11, 12]. Despite these efforts, there remains a gap in understanding the specific causes or contributors to concept drift, particularly in the domain of malware detection and network intrusion detection. Addressing this gap, our thesis seeks to distinguish between two primary forms of concept drift, feature-space drift and data-space drift. We assess their respective impacts on changes in model behavior. By isolating and measuring the influence of these types of drift, we aim to provide a nuanced understanding of concept drift's underlying causes, thereby informing the design of more resilient ML-based security detectors in the future.

Consider the case of an ML model developed specifically for the detection of ransomware. To illustrate the distinction between feature-space and data-space drifts, let us refer to Figure 1.1. In the scenario depicted in the top figure, we observe an instance of data-space drift. The model relies on a predefined set of two features for its operation: f_1, representing the "number of writes" to disk, and f_2, quantifying the "complexity of the call graph." This set of features remains constant over time. At an initial time point, t_1, the model's decision boundary is represented by a red dashed line. As time progresses to t_n, both ransomware and

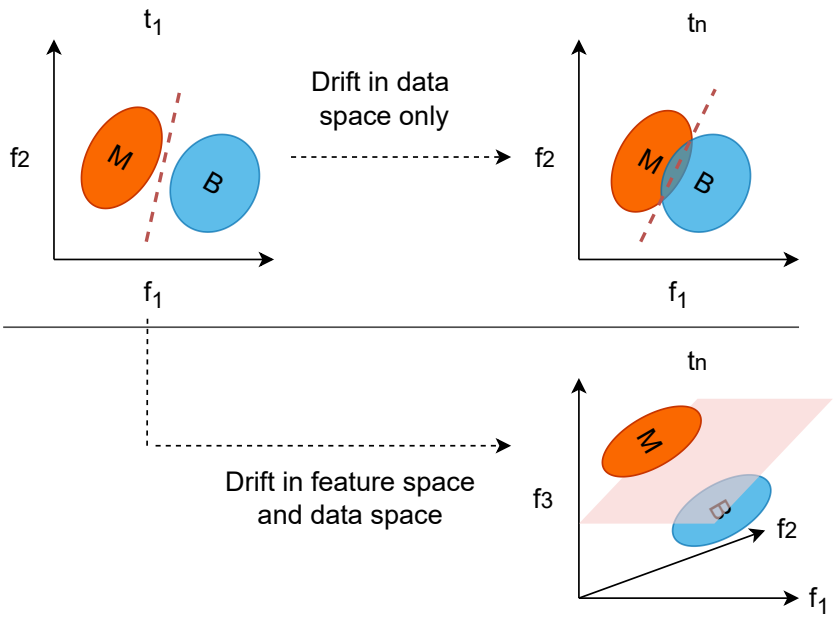

Figure 1.1: Motivation Example.

benign programs exhibit shifts in their behaviors concerning these two features, potentially leading to a modification of the decision boundary and, consequently, to classification errors.

In contrast, the bottom figure exemplifies feature-space drift. Rather than adhering to a static feature set, the model is designed to incorporate new features in response to emerging threats. For instance, contemporary ransomware may begin utilizing a novel cryptography library, denoted as "X." In response, a new feature, f_3 ("presence of crypto-lib-X"), is added to the model's feature space at time t_n. With the expanded feature space (f_1, f_2, f_3), it becomes possible to identify a hyperplane that effectively separates the two distributions, illustrating the benefit of adapting the feature space to accommodate new information. This concept forms the basis of online learning approaches in malware detection, where the feature set is dynamically updated to address the challenges posed by concept drift [11, 12, 13].

By providing a more detailed exploration of concept drift and its implications for ML-based security detection, along with a nuanced example illustrating the differences between feature-space and data-space drifts, this extended text aims to offer a comprehensive understanding of the phenomenon and its significance in ensuring the effectiveness and reliability of detectors over time.

Our Work: In our thesis, we conducted two technical experiments to advance the understanding of the reason and impact of concept drift in the realm of malware detection and network security.

Firstly, we undertook an exploration of both feature drift and data drift using datasets

pertinent to Portable Executable (PE) files and Android applications. This exploration aimed to isolate and measure the impact of these two types of drift on the behavior of machine learning models, particularly those used in detecting malware. By analyzing the shifts in feature and data distributions within these datasets, we sought to uncover insights that could inform the development of more adaptive and resilient detection systems. Both datasets are from real-world industry environments.

Secondly, we endeavored to design a model specifically tailored to detect concept drift within network traffic, utilizing the Network Security Dataset from a real-world SOC (Security Operations Center) as a case study. Our approach involved the innovative use of cross-host features combined with Support Vector Machine (SVM) techniques to accurately identify shifts in data patterns. The design of this model was guided by the hypothesis that detecting concept drift in network environments is critical for maintaining the effectiveness of cybersecurity measures, especially in the face of evolving threats. The comprehensive network traffic dataset from the SOC is distinguished by its real-world origins and extensive coverage over a prolonged period. This dataset is invaluable, encapsulating numerous verified attacks alongside many false alarms. We also do a measurement on this dataset to unravel the intricacies of various real-world network alerts, enhancing our understanding of their characteristics and behaviors.

By addressing the reason and impact of both feature-space and data-space drift in distinct contexts, our work sheds light on the complexities of maintaining the efficacy of malware detectors and network intrusion detectors in dynamic environments. The insights gained from our study are intended to pave the way for future research and development efforts aimed at creating more adaptive, reliable, and effective security mechanisms in the ongoing battle against cyber threats.

Main Contributions:

- **First**, we investigate the effects of feature-space drift versus data-space drift on the performance of malware detectors using two real-world malware datasets. Feature-space drift refers to changes in the features used to detect malware, while data-space drift involves changes in the underlying malware data without altering the feature set. Our experiments are designed to compare the impact of these two types of drift on common malware detection systems. Our findings reveal that feature-space drift has a minimal impact on detection performance compared to data-space drift, where the feature set remains constant. This suggests that current malware detectors are more significantly challenged by shifts in the data distribution itself rather than changes in

the feature landscape. These results highlight the importance of developing malware detection systems that can adapt to data-space drift, ensuring sustained detection effectiveness in evolving threat environments.

- **Second**, we perform empirical measurements on a real-world network traffic dataset from an anonymous SOC. we explore how alerts are fired by the network monitoring tool and answer the following questions: How often are the alerts associated with malicious activities? To what extent can the SOC analyze and take action on these alerts automatically? What are the common reasons behind the fired alerts and the challenges to reasoning the alerts?

- **Third**, we introduce a novel approach to the detection of concept drift in a network traffic dataset obtained from a real-world Security Operations Center (SOC). We want to answer the following questions: Does data space drift exist in network data? How to build more robust features against data space drift in network attack detectors? Recognizing the limitations of traditional feature engineering techniques in capturing the complex and evolving nature of network traffic, we propose an advanced feature engineering methodology that synthesizes both host-level and cross-host features. Our approach is distinguished by its emphasis on cross-host features, which leverage the relationships between different logs to enhance the robustness of the model against concept drift. By integrating these features, we construct a more comprehensive representation of the network traffic, enabling the Support Vector Machine (SVM) model to detect data space drift. We prove the existence of data space drift in network data, and the inclusion of cross-host features marks a significant advancement in the field, offering a potent mechanism to counteract the challenges posed by data space drift.

CHAPTER 2: BACKGROUND AND RELATED WORK

2.1 CONCEPT DRIFT IN MACHINE LEARNING (ML).

Concept drift is a general challenge for ML models. After a trained model is deployed, the testing data distribution may (gradually) shift away from that of the training data over time. Such concept drift can lead to model degradation [6, 14]. To detect concept drift, ML researchers have proposed various methods to statistically assess model behaviors and measure distribution changes [15, 16, 17, 18]. However, most of these methods require collecting extensive labels for the new data, which can be challenging for security applications.

2.2 CONCEPT DRIFT IN MALWARE DETECTION.

Machine learning has been used for malware detection [1, 2, 3, 4, 5] and malware family classification [19, 20]. Concept drift also poses a challenge to these models, which requires the models to be periodically re-trained [21, 22]. Recent works propose to proactively detect drifting samples and use a smaller set of drifting samples for model updating [7, 8, 9, 10], which can reduce the overhead of data labeling.

2.3 CONCEPT DRIFT IN NETWORK DATA.

Concept drift is particularly prevalent in network traffic data, which is inherently non-stationary, exhibiting a dynamic nature as attackers continuously develop new strategies to evade detection. Traditional machine learning models, including Deep Neural Networks (DNNs), often operate under the assumption that data distributions are stationary, an assumption that falls short in the face of the polymorphic and evolving nature of malicious activities within network environments. The work by Sommer and Paxson [14] initially highlighted the unique challenges in applying machine learning to intrusion detection, noting the domain's particular characteristics that complicate the deployment of these technologies. Despite advances, the issue of concept drift continues to plague network intrusion detection systems (NIDS), rendering them less effective as they struggle to adapt to new, unseen attack vectors. Recent studies and frameworks, such as INSOMNIA [23], have begun to address these challenges directly, proposing innovative solutions like semi-supervised learning and active learning strategies to mitigate the impact of drift and improve the adaptability and performance of NIDS in ever-changing network environments. These efforts underscore

the critical need for NIDS to evolve beyond static models, embracing mechanisms that can dynamically adapt to concept drift and maintain their efficacy over time.

2.4 FEATURE-SPACE VS. DATA-SPACE DRIFT.

While most existing works focus on concept drift detection [7, 8, 9, 10], less effort is investigated to *reason the causes of the drift*. We focus on *feature-space drift* and compare it with *data-space drift*. For data-space drift, suppose a malware classifier uses a fixed set of features F, concept drift happens when the data distribution $D(F)$ changes over this feature set. Feature-space drift additionally considers the changes in the underlying feature space ($F \rightarrow F'$) which then leads to new data distribution ($D(F')$). Note that feature-space drift always leads to data-space drift because data distribution $D(F')$ is dependent on feature space F'.

In the context of malware detection, feature-space drift describes the situation where new samples introduce new features (e.g., APIs, strings, libraries) that are never seen in the previous data. When such features appear, the model can either choose to ignore them or incorporate them. For example, CASANDRA [13] and DroidEvolver [11] adopted online learning algorithms to incrementally add features to the feature space as new samples arrive. Another recent system APIGraph [24] proposes to enhance this process by learning a robust feature space to reduce the frequency of feature updating. Our work is built on top of these existing works with a focus on (a) empirically comparing the impact of feature-space drift and data-space drift and (b) exploring the reasons behind the observed drift.

2.5 SECURITY OPERATIONS CENTER (SOC).

SOC is an in-house or outsourced unit responsible for monitoring information technology (IT) infrastructures and responding to security incidents [25, 26]. SOCs have three important aspects: people, processes, and technology [26, 27]. First, people: SOCs have a team of security professionals with various duties and roles (e.g., threat hunters, security analysts, and team managers). Second, processes: SOCs involve various workflows in their routine tasks such as vulnerability scanning and handling alerts from Intrusion Detection Systems (IDS) or Security Information and Event Management (SIEM) systems. SOCs typically maintain a *playbook* [28] to document a standard procedure for communicating about and escalating alerts. Third, technology: SOCs may deploy various technical tools to monitor the network and hosts to detect attacks [26, 29, 30, 31, 32, 33].

2.6 USER STUDIES WITH SOC MEMBERS.

Researchers have studied various challenges faced by SOC analysts by conducting *user studies*. Existing studies have explored SOC analysts' perceptions of IDS usage [34], effectiveness of playbooks [28], system misconfigurations [35], malware detection strategies [36], SOC performance evaluation metrics [36], the "burnout" issues of analysts [37], and how analysts resolve contradictions between people and tools [38].

In particular, Kokulu et al. [25] conducted an interview with 18 SOC members who expressed concerns about the high volume of unfiltered/uncorrelated alerts. However, surprisingly, participants were not concerned about "false positives" (FPs) of the detection tools. Alahmadi et al. [26] further interviewed 20 SOC members to explore the reasons. They revealed that a common perception is that the majority of the "false positives" are caused by *benign triggers*. Benign triggers are correctly fired alerts but can be explained by legitimate behaviors in the organization's context (and thus are safe to ignore).

Our study complements and advances existing literature (which are primarily qualitative user studies) by providing a *quantitative* lens into the problems of security alerts and attack investigation using real-world SOC logs. Also, with empirical data across multiple years, we provide a longitudinal view of how these problems evolve over time.

2.7 SOC TOOLING.

Researchers have proposed technical solutions for threat detection and forensics for SOCs. These solutions broadly include network intrusion detection systems (NIDS) [7, 39, 40, 41, 42], host-based IDS [43, 44, 45, 46, 47, 48], alert triangle [49, 50, 51, 52], verification [53], correlation [30, 54, 55, 56, 57] and prioritization [58, 59] methods, provenance-based threat hunting systems [60, 61], and attack prediction models [62, 63]. The goal of our work is not to develop specific tools but to use real-world data to understand the gaps between existing tools and the needs of SOCs.

CHAPTER 3: CONCEPT DRIFT IN MALWARE DETECTION

Machine learning (ML) models are increasingly used for malware detection, facing a significant challenge: concept drift. This occurs when the characteristics of data change over time or due to adversaries' efforts, leading to decreased model performance. Previous research has focused on identifying and managing these changes but has not deeply explored the reasons behind concept drift. We address this gap by examining two main causes: feature-space drift and data-space drift. By understanding these factors, we aim to enhance future malware detection models to better resist concept drift.

Research Question:

- How does feature drift versus data drift affect the accuracy and reliability of machine learning-based malware detectors over time?
- Between feature drift and data drift, which has a more significant impact on the adaptability of machine learning algorithms used in malware detection, and how can this knowledge inform the development of more robust detectors?

Disclaimer: The research work in this chapter has been published in a research paper at DLSP [64] (me as the second author, with other co-authors including Z. Chen, Z. Kan, L. Yang, J. Cortellazzi, F. Pendlebury, F. Pierazzi, L. Cavallaro, and G. Wang).

3.1 METHOD

The goal of our first technical experiment is to decouple the impact of feature space drift with that of data space drift to understand the main contributor of concept drift. To do so, we take a malware dataset and divide it into n timestamped data blocks $[D_{t_1}, D_{t_2}, ..., D_{t_n}]$. Here, "data block" is just a general concept. In actual experiments, they can be generated with a non-overlapping data split or using a sliding time window.

1. **Measure data-space drift only**: we construct a feature space F_1 based on the first data block D_{t_1}. Then we fix this feature set F_1 for all the following data blocks. This feature set will be used for all the model training/re-training and testing for the following data blocks.

2. **Measure feature-space drift**: for a given data block at t_i, the corresponding feature space F_i will be updated according to the recent data block. The corresponding classifier training/re-training at t_i will be based on F_i.

Types of Features. Malware detection models use different types of features. For our analysis, we categorize features into two types: (1) Data-dependent features—new features can be introduced as new data arrives. For example, for PE malware, such data-dependent features can be the set of *imports*, *exports*, *libraries*, and *dll* (carried in new malware samples). (2) Data-independent features—this type of feature is universally applicable to all samples. For example, features such as *file size* and *number of sections* do not need to be introduced by new data and can be computed for any PE files. For our analysis, feature-space drift is mainly considering data-dependent features.

3.2 ANALYSIS: ANDROID MALWARE

Dataset. We construct an Android malware dataset sampled from AndroZoo [65] between January 2015 and December 2021. We take the 2015–2016 data from [66] and 2017–2018 data from [9]. Then to obtain more recent data, we sampled APKs from AndroZoo between 2019–2021 to combine with existing datasets. We follow the recommendation from [22] to maintain the temporal and spatial consistency for the evaluation. For each month, we have at least 2,000 samples and maintain a malicious-to-benign ratio of 1:9.

Following prior works [22, 67], an APK is labeled "benign" if no VirusTotal engine has flagged it as malicious; an APK is labeled "malicious" if at least four engines have labeled it so. The rest of the grayware is excluded. In total, the dataset contains about $32,000$ malware samples and $279,000$ benign samples from 2015–2021.

We used Drebin [1] to extract feature vectors from these Android apps. Drebin has 10 feature categories such as *Activities*, *URLs*, and *APIs*. Each feature is treated as a string entity. Given an app, Drebin produces a binary feature vector of values of 1 or 0—"1" means the app contains this feature (e.g., an API), and "0" means the app does not contain this feature. According to our definition, all of the features are "data-dependent" since new features (e.g., an API that never appeared before) can be introduced as new data arrives, which are subject to feature-space drift.

We follow a common approach [68] to performing feature selection before training a detector (to improve training efficiency). We use the LinearSVM L_2 regularizer to select the top 20K features and train an MLP binary classifier with one hidden layer of 1,024 neurons and a dropout rate of 0.2. We use an MLP as the default model because it is a commonly

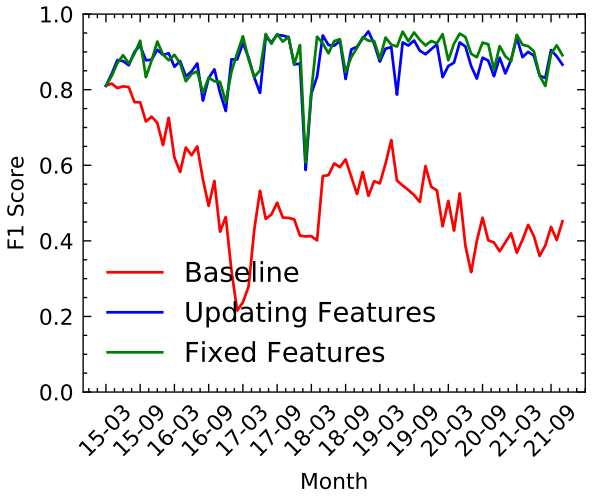

Figure 3.1: Training data ratio: 100%

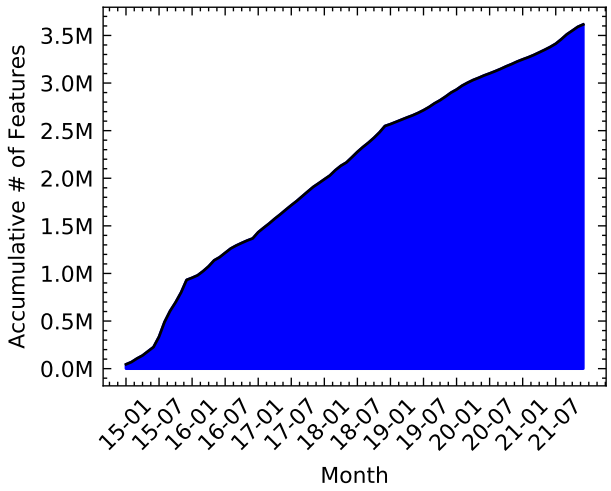

Figure 3.2: **Cumulative # of features**—This analysis considers all the features that appear in the dataset, before running any feature selection.

used model for this problem [4, 69, 70, 71]. We will also test SVM and SecSVM [68] for comparison.

Experimental Setup. We divide the data between 2015 and 2021 into 84 months and construct three settings to measure feature-space and data-space drifts.

First, we construct a *baseline* to measure the level of concept drift when the model is never updated after training. We use the first three months of data for training and then test it in the remaining 81 months without any re-training.

Second, we construct a *fixed feature space* setting. The idea is to always keep the same feature set so that we can exclusively measure the data-space drift. We extract the features during the first three months and then fix this feature set for the entire experiment. It means no new features will be added/removed during the subsequential model training and testing. Here, we re-train this model every month using the data in a sliding window of the past three months. After re-training, we test the updated model in the next month. For example, at the end of month 3, we will use data from months 1, 2, and 3 as the training set to retrain the model. We then test this model in month 4 to report the F1 score. This ensures there is no data leakage between training and testing. This process repeats at the end of month 4. Note that we assume abundant labels are available to study the upper-bound performance. In practice, there are various techniques [7, 8, 9, 10] to use a smaller number of labeled samples for retraining, which is not the focus of this thesis.

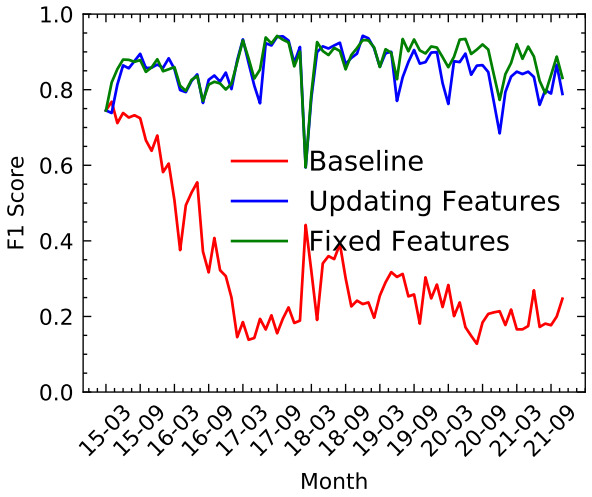

Figure 3.3: Training data ratio: 50%

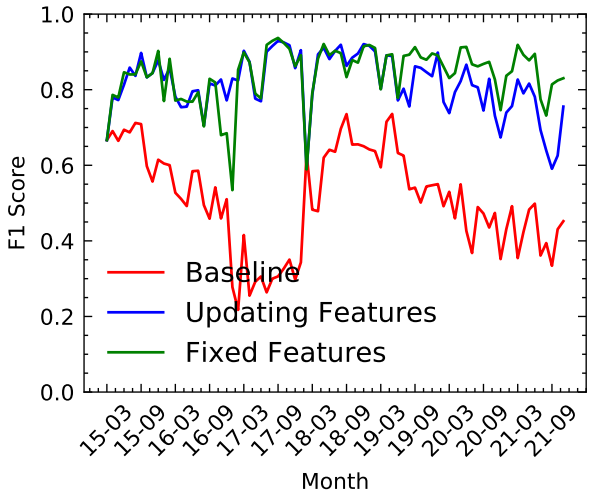

Figure 3.4: Training data ratio: 25%

Third, we construct an *updating feature space* setting to examine the impact of the feature-space drift. This setting is similar to the second setting above, except new features from the recent sliding window can be included for retraining. During each re-training, feature updating is done by using a LinearSVM L_2 regularizer to select the top 20K features based on the data in the sliding window. As discussed in 2, feature-space drift always affects the data space because data distribution $D(F)$ is dependent on the feature set F.

Drift in the Android Dataset. As shown in Fig. 3.1, the baseline (red line) confirms the existence of concept drift as the model performance (F1 score) is decreasing over time. To counter this effect, we show that re-training helps (both blue and green lines) as the model performance can be maintained at a high level after re-training.

Surprisingly, the blue line and the green line almost overlapped. Recall that the blue line represents the setting where features are updated monthly using recent data, and the green line represents the setting where the features are fixed. This result indicates the model performance is not benefiting from the feature-space update. The fixed feature set (20K features) initially selected in the first three months of 2015 still works well later in 2021. This also indicates that data-space drift (over existing features) is the dominating cause of model performance degradation over time.

To better interpret the result, we revisit the toy example in Fig. 1.1. The result means introducing the new feature f_3 (*crypto-lib-X-is-present*) to the feature set is not as critical as updating the data distribution over existing features (f_1 and f_2). A possible reason is that the behavior shift can be readily captured by existing features, e.g., ransomware now does 10x more writes than before, which can be captured by the existing f_1 (*number-of-writes*)

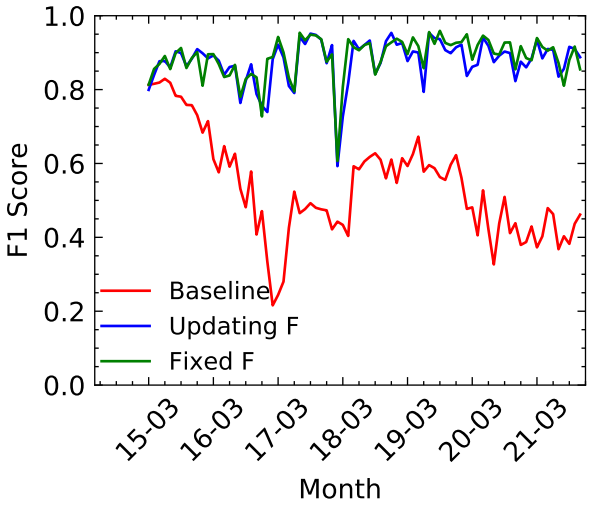

Figure 3.5: Feature space size: 10,000

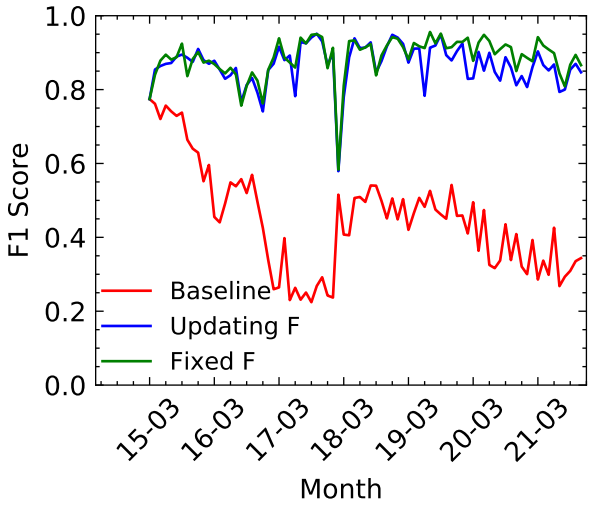

Figure 3.6: Feature space size: 50,000

during re-training.

We also observe the green line gets a bit higher than the blue line during the later years. A possible explanation is that frequently updating feature space may take in features that are only temporally useful (i.e., due to short-term drift), but can hurt the models' long-term performance.

Impact of Training Data and Feature Space Size. We further explore under which condition feature-space drift starts to have an impact. In Fig. 3.3 and 3.4, we first try to reduce the data used for training and re-training. The rationale is that, with less data, it might be more difficult to capture concept drift, and thus the model is more dependent on a good/updated feature space to perform well. This also mimics a real-world scenario where labeled data is limited. The result, however, shows that using less training data does not separate the blue and green lines.

Next, we further explore the impact of feature set size, i.e., the number of selected features. The default feature set size is 20K features. In Fig. 3.5 and Fig. 3.6, We further test 10K and 50K. We find that using a larger or smaller feature set does not make a notable difference.

We have also tested the impact of ML models (using SVM and SecSVM) and did not observe major differences between the blue and green lines.

Feature Space Analysis. To understand the reasons behind our observation, we further analyze the feature space. A natural question is, is it because there are no major feature changes over time? Fig. 3.2 suggests the answer is "no". If we consider all the features (without feature selection), there are about 100K unique features observed during the first

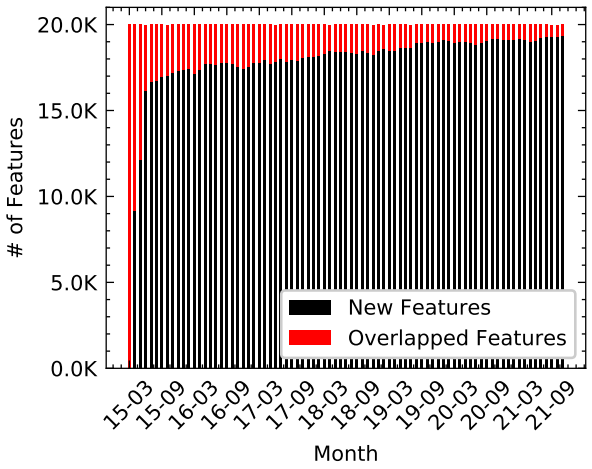

Figure 3.7: **Updating feature setting**—Number of overlapped features in each training window compared with the initial feature set (first 3 months).

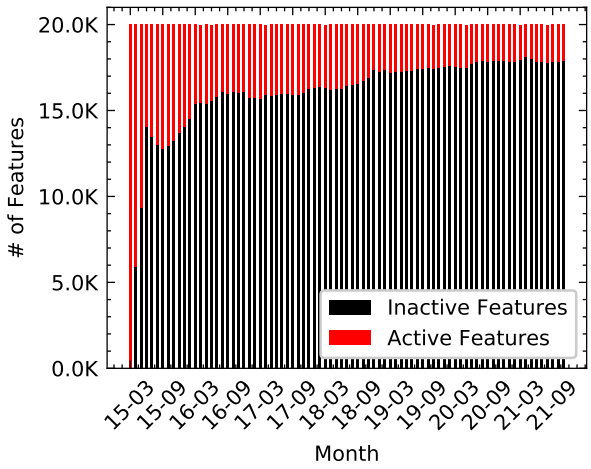

Figure 3.8: **Fixed feature setting**—Number of active and inactive features in each sliding window. "Active" means at least one sample has this feature.

three months, and 3.6 million features by the end of 2021—a significant number of features are introduced over time.

Then after feature selection, the total number of features is capped at 20K, and thus the level of feature dynamics is reduced. For the "updating feature" setting, Fig. 3.7 shows the number of features in each sliding window that are overlapped with the first three months (i.e., the initial feature set). The black bar (new features) gets bigger while the red bar (old features) gets smaller over time. This shows new features are indeed selected during periodical retraining.

We further examine the features under the "fixed feature" setting in Fig. 3.8. For each sliding window, we show the old features that are still active (i.e., appeared in at least one sample in the window) and inactive features (i.e., did not appear in any samples). The result indicates that about 2.1K features remain active at the end of 2021. We further analyze the importance score (produced by the feature selection method) and confirm these features have a higher average score (0.054, STD=0.075) than the inactive features (0.032, STD=0.060). This indicates that a small set of important features (selected in the beginning) is sufficient to maintain the model performance over multiple years, without any feature updating.

3.3 ANALYSIS: PE MALWARE

We validate our observations with PE malware detectors.

Dataset. We use the EMBER PE dataset [3] that contains over 2 million PE files collected between January 2017 and December 2018. The dataset includes 800K malicious, 700K benign, and 500K unlabeled files. For this experiment, we only use labeled malicious and benign files.

EMBER extracts three types of features from a PE file, namely, numerical, boolean, and string features. Numerical features record the numerical value of certain properties of the file (e.g., *file size*, *number of sections*). Boolean features record the binary value ("yes" or "no") such as *has_debug* and *has_signature*. String features encode discrete entities such as the set of *imports*, *exports*, and *libraries* in the PE files. There are 10 types of such strings. According to our definition in 3.1, only the string features are "data-dependent" since new features (e.g., libraries that never appeared before) can be introduced as new data arrives. Numerical and boolean features are pre-defined (which are not introduced by the new data).

For string features, EMBER proposes to use locality-sensitive hashing (LSH) to convert/embed a set of string entities into a fixed-length vector. For this experiment, we use MinHash [72] for feature embedding which preserves the similarity between the string sets in the embedding space.

In the EMBER dataset [3], string-based features are the main source of feature-space drift, and we provide more context for their embedding method. There are ten categories of string features including *section_name*, *section_characteristics*, *library*, *import*, *export*, *machine*, *subsystem*, *dll_characteristics*, *header_characteristics*, and *magic*. For each file, the raw features of each category are presented as a set of discrete strings/entities. Given a large number of unique entities, we follow EMBER [3] to embed the string features using a hashing function that maps a set of strings to a fixed-length vector. We choose the vector length for each string category based on the suggestion of the original paper [3]. We use a locality-sensitive hash (LSH) method—the advantage is that LSH preserves the similarity between sets, i.e., if two sets are more similar (i.e., with a bigger intersection), their hashed vectors also have a smaller distance in the embedding space. Hashing helps to map a sparse feature space with potentially tens of millions of features into fixed-sized dense vectors (e.g., size of 2,381). This helps to improve the efficiency of training. Note that the original EMBER paper [3] used FeatureHasher in the sklearn library which is an LSH with respect to cosine distance. We used MinHash [72] for easy implementation, and we confirmed that both hashing algorithms have equally good performances (i.e., $< 1\%$ difference in F1 score).

Experimental Setup. We follow the original EMBER paper [3] and use LightGBM to train the malware detector for its high efficiency and good detection performance [73]. We divide the data between 2017 and 2018 into 24 months and test the three settings described

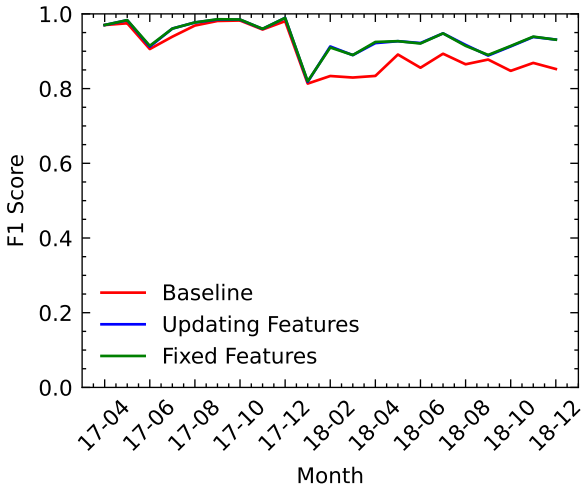

Figure 3.9: PE results (all features).

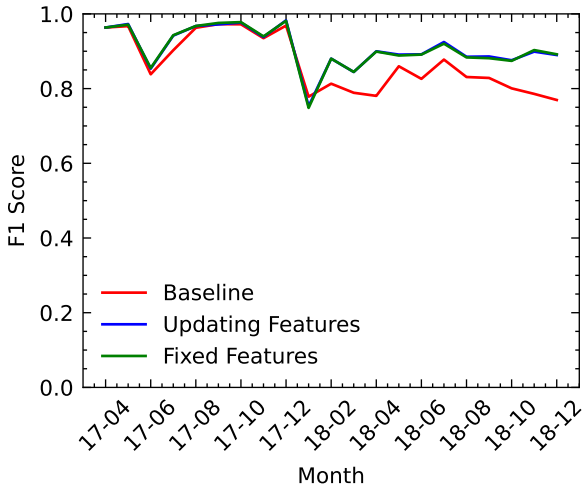

Figure 3.10: PE results (string features).

in 3.2: (1) baseline setting, (2) fixed feature space, and (3) updating feature space. Note that settings (2) and (3) are only applied to the *string features* since these features are data-dependent (i.e., subject to feature-space changes).

Drift in the PE Dataset. As shown in Fig. 3.9, the baseline (red line) confirms that concept drift exists, given the clear drop in the F1 score after the first month of 2018 (dropped by 17%) and stays at the lower level. This result echoes the EMBER paper [3] as the authors intentionally included "harder-to-detect" malware in the 2018 data.

In our analysis, we consistently notice a significant overlap between the blue and green lines, underscoring that the outcomes of the two distinct settings are remarkably similar. This pattern strongly implies that the degradation of model performance is primarily driven by data-space drift. In other words, changes in the underlying data over time seem to be the main factor diminishing the model's accuracy and effectiveness. On the other hand, updating the model based on feature space—altering the model to better understand or represent the features of the data—appears to have minimal influence on mitigating this degradation. This finding is crucial for our understanding of model maintenance and improvement strategies. It suggests that to preserve or enhance model performance over time, efforts should be more concentrated on addressing data-space drift, perhaps through techniques like regular retraining with new data or employing adaptive learning methods that can dynamically adjust to changes in the data landscape.

Models with String Features Only. An alternative explanation might be that the features subject to drift (i.e., string features) did not play a major role in the model. To

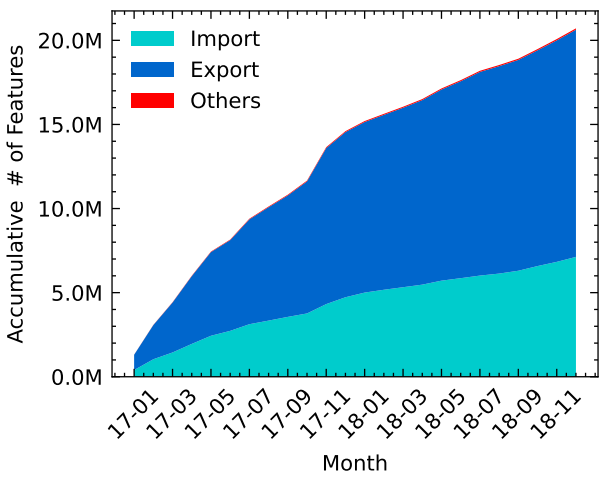

Figure 3.11: We show the number of unique features over time for three different types: Import, Export and Others over 2 years.

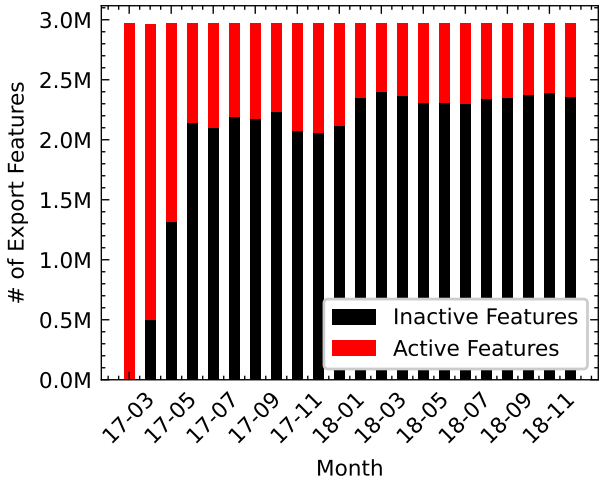

Figure 3.12: "Fixed feature" setting: we show the number of active and inactive *export* features in each sliding window.

eliminate this possibility, we perform the same experiment by using *string features only*.

As shown in Fig. 3.10, when only using string features, the conclusion remains the same. The overall performance drops slightly compared to using full features (Fig. 3.9). However, the blue and green lines still overlapped.

Feature Space Analysis. To show there is indeed a high level of feature dynamics, we plot Fig. 3.11, which shows the accumulative number of raw string features (before hashing) over time. We note that *import* and *export* contributed to the majority of the unique strings. We can also observe a sudden increase in the number of new features in early 2018, which explains the sudden performance drop during that time (Fig. 3.10). The result confirms that a large number of new features (strings) were introduced, from 4.4 million in the first three months to 20.7 million by the end of 2018.

Despite the high level of feature dynamics, the classifier's performance is not affected, possibly due to feature embedding. The hashing function (LSH) has helped to map a large, sparse feature space (i.e., 20.71 million strings) into a fixed-sized dense feature space (in our case, the hash vector length is 2,381), which can stabilize the feature space. Another reason is that the initial feature set (first three months) is still quite actively used by samples in the last month of the two-year period (see an example in Fig. 3.12). These features remain effective in separating malicious from benign samples (F1=0.93, Fig. 3.9).

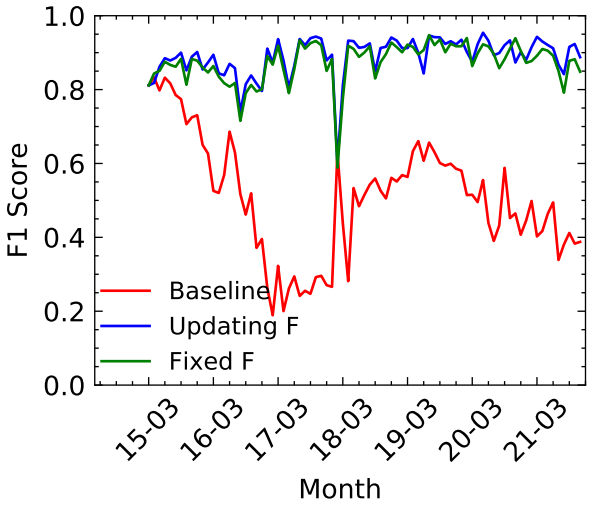

Figure 3.13: Model: SVM

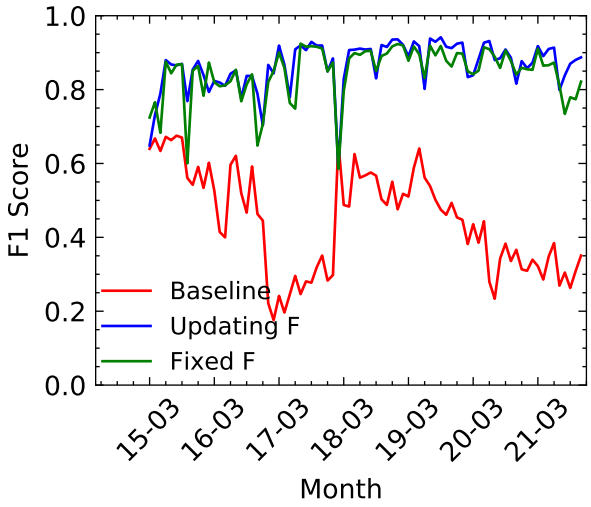

Figure 3.14: Model: SecSVM

3.4 IMPACT OF DIFFERENT MODELS

In this section, we evaluate the impact of model choices on our experiment result on the Android dataset. In addition to the MLP model (used in 3.2), prior works have also used LinearSVM and SecSVM for Android malware detection [67]. Note that SecSVM [68] is a variant of the SVM model designed to be more robust against evasion attacks. The main idea of SecSVM is to maintain more evenly distributed feature weights such that the model cannot be easily evaded by manipulating a few features. We follow the same experimental methodology from 3.2, and test LinearSVM and SecSVM respectively under the default setting (20K feature set size, 100% training data).

The results are presented in Figure 3.13 and Figure 3.14. We find that our conclusion remains the same.

With LinearSVM and SecSVM, the blue and green lines are still largely overlapping, indicating that data-space drift is still the dominating contributor to concept drift.

3.5 DISCUSSION

In this study, we delve into the distinct roles of feature-space drift and data-space drift to ascertain their respective impacts on the efficacy of malware detection systems. Our empirical investigations reveal a notable trend: data-space drift emerges as the primary factor contributing to the decrement in model performance over time, whereas the influence of updating the feature space appears minimal, if not negligible. This observation holds across a spectrum of feature types and engineering techniques, as well as across malware

detectors designed for both Android and Portable Executable (PE) files.

The implications of these findings are significant. They suggest that a robust and well-designed feature set is capable of underpinning model re-training efforts aimed at addressing data-space drift, without necessitating modifications to the feature set itself. This remains applicable even in the face of a substantial influx of new features introduced by newly analyzed samples.

However, it would be premature to conclude that updating features is entirely redundant. The nuanced dynamics between feature-space stability and model adaptability call for a more thorough exploration to reach a definitive understanding.

In light of these considerations, it becomes clear that a deeper understanding of the implications of feature-space updating is crucial. Further research in this area is essential to develop strategies that optimize the balance between feature stability and the need for models to remain responsive to evolving malware landscapes.

CHAPTER 4: CONCEPT DRIFT IN NETWORK INTRUSION DETECTION

In the contemporary landscape of cybersecurity, the proliferation of sophisticated threats poses significant challenges to safeguarding digital assets and ensuring the integrity of network infrastructures. In this context, the analysis of real-world network traffic datasets assumes paramount importance in unveiling the intricate patterns of attack vectors and understanding the evolving nature of cyber threats. On the other hand, Security Operations Centers (SOCs) face the key challenge of handling excessive security alerts. While existing works have studied this problem qualitatively via user studies, there is still a lack of *quantitative understanding* of the impact of excessive alerts and their effectiveness and limitations in capturing true attacks and compromises. In this technical experiment, we delve into an examination of a network traffic dataset obtained from a real-world Security Operations Center (SOC). **Firstly**, we conduct an in-depth measurement of network alerts and attack attempts within the dataset to discern patterns of malicious activity. **Secondly**, we measure the concept drift, particularly the data space drift, in-network data from that SOC. Our framework includes a novel feature engineering method (including the exploration of cross-host features) and an SVM-based model. Through rigorous experimentation, we substantiate the existence of data space drift and demonstrate the efficacy of incorporating cross-host features to enhance the robustness and reliability of the detection model. This research aims to shed light on the dynamic nature of cyber threats and advance the development of proactive defense mechanisms to fortify network security. It's important to clarify why we are not focusing on feature-space drift in network data. In the context of network traffic, the feature set is typically fixed. This differs from scenarios such as malware detection, where new features are constantly being introduced by new malware samples or benign software. In network traffic analysis, the focus is primarily on understanding shifts in the distribution of data rather than on changes in the feature space itself.

Research Question:

- How often are the alerts associated with malicious activities? To what extent can the SOC analyze and take action on these alerts automatically? What are the common reasons behind the fired alerts and the challenges to reasoning the alerts?
- Does the concept drift, especially the data space drift, exist in real-world SOC network traffic? Which features are more important and efficient in building models to detect and resist such concept drift?

Disclaimer: The research work in this chapter has been included in a research paper that is still in the Major Revision phase (me as a joint author, with L. Yang, Z. Chen, C. Wang, S. Booma, P. Cao, C. Adam, A. Withers, Z. Kalbarczyk, R. Iyer and G. Wang).

4.1 DATASET DESCRIPTION

We obtain data via collaboration with a Security Operations Center (SOC) within a supercomputing center for scientific research. The name of the organization is anonymized. The center has thousands of physical nodes, tens of thousands of computing cores, and 10+ petabytes of storage interconnected internationally via a terabit network.

The SOC monitors the network/system and user activities leveraging a wide spectrum of internal, open-sourced, and commercial tools to detect and respond to security incidents. Figure 4.1 shows a simplified view of the process. The SOC analysts use a Splunk dashboard for Security Information and Event Management (SIEM) [74], which provides a real-time view of security events collected by network monitoring tools, system logs, Linux kernel audit logs, and other internal and external data sources (e.g., blocklists, honeypots, threat intelligence).

Network Alert Dataset (2018–2022). Our preliminary dataset contains the *network-level* alerts fired between April 2018 and July 2022 by the Security Operations Center (SOC)'s network monitoring tool Zeek (a widely-used open-source tool [75]). As shown in Figure 4.1, the SOC maintains a customized rule engine that analyzes a variety of network and system logs to detect events that match the pre-defined rules. The network monitoring tool is one of the main sources of alerts. There are other alerts from endpoint/host-IDS, partner sites, and other commercial tools, to which we don't have access. In other words, the dataset represents a lower bound of alerts from this SOC.

In total, the dataset contains 115,617,526 network-level alerts. Each alert is characterized by a timestamp, involved IPs, an alert category, and other metadata. Starting in 2020, the SOC implemented a Black Hole Router (BHR) [76] (see Figure 4.1) to automatically handle a subset of these Zeek alerts without human involvement (e.g., blocking IPs associated with the alert) in real-time. As such, our dataset contains the BHR label from 2020 to 2022.

Ground-Truth: True Attacks. To understand how effective these network alerts are in informing the Security Operations Center (SOC) of true attack incidents, we also gather data about the ground truth of successful attacks and compromises in the past two decades. This dataset is from "post-attack forensics", as illustrated in Figure 4.1. The definition

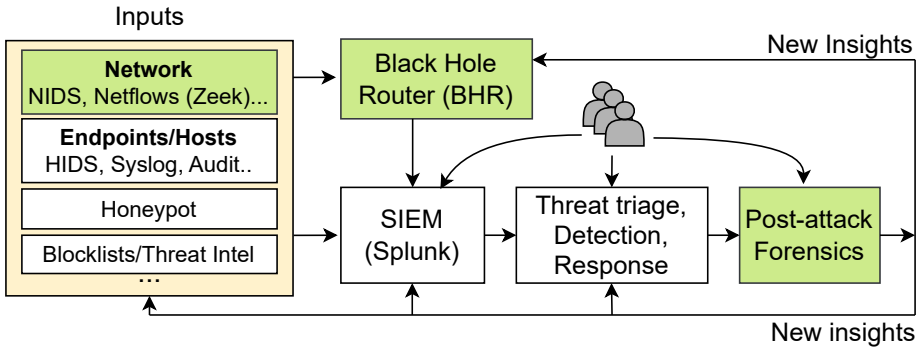

Figure 4.1: **SOC Workflow**—The highlighted green boxes are the main data sources for our thesis. SIEM stands for "Security Information and Event Management."

of *true attack* (see Table 4.1) is based on whether the attacks have caused major degradation/interruption to system operation or whether there was evidence of data leakage or system compromise. This dataset contains 227 true attack incidents from January 2002 to March 2022. Among them, 11 occurred during the overlapping period with the alert dataset (2018–2022). For each incident, SOC analysts collected a set of attack artifacts (e.g., logs, binaries, VM images), conducted a detailed investigation, and produced a report describing their investigation process and conclusions. The reports were written in an unstructured format.

4.2 A MEASUREMENT OF NETWORK ALERTS

In this section, we explore how alerts are fired by the network monitoring tool and answer the following questions: How often are the alerts associated with malicious activities? To what extent can the Security Operations Center (SOC) analyze and take action on these alerts automatically? What are the common reasons behind the fired alerts and the challenges to reasoning the alerts?

4.2.1 **Complex Reasons for Triggering Alerts**

We use the alert datasets to understand the reasons why alerts are triggered. For this analysis, *we focus on the second period (2020–2022)*, because (1) the rule sets are more optimized to reduced alerts; and (2) only the second period has auto-blocking BHR, which is important for identifying attack attempts. At a high level, we categorize the reasons into several categories (Table 4.2). We use the same definitions presented in Table 4.1.

Term	Definition
True attack incidents	The attacker successfully compromised the system and/or caused damages
Attack attempts	The attacker tried but failed to compromise the system or cause damages
False positives	Benign activities that are incorrect determined as security attacks
Benign triggers	Correctly matched security events but have business-justified explanations [26]

Table 4.1: **Definitions**—Key terms used in the thesis.

True Attacks. True attacks refer to incidents where the attacker successfully compromised the system and/or caused damages. Since successful attacks are rare, the number of alerts is also small (1,114, 0.01%).

Attack Attempts. Attack attempts are those that (1) were initiated by malicious attackers, but (2) did not manage to compromise the system or cause damage. For our analysis, we rely on the auto-blocking mechanism (BHR) to flag ground-truth attack attempts. Recall that BHR is designed to be "conservative" to only block attack attempts that match high-confidence signatures/patterns. Through our analysis, we find that not all the BHR alerts represent attack attempts. In total, 44.9% of the alerts are BHR-blocked—after excluding benign triggers in them (about 17.50%), we located 27.37% alerts that are associated with attack attempts (see §4.2.2).

Benign Triggers. Alerts that are neither "true attacks" nor "attack attempts" cannot be simply marked as false positives (FP). Many of them can be benign triggers. Benign triggers [26] are *correctly fired alerts*, explained by business-justified, acceptable behaviors in the organization's environment. As such, analysts may choose to ignore them. An example would be an alert fired correctly because a weak SSL key is detected, but SOC can ignore the alert knowing the host is retiring and disconnected from the public network.

Unknown. The remaining alerts (23.72%) are marked as "unknown" since we cannot easily/confidently determine their causes. They could be a mixture of attack attempts, unknown benign triggers, false alerts, or even true attacks that are missed by the SOC team. We will further analyze them by statistically comparing them with other alert categories, especially true attack alerts.

Alert Type	# of Alerts	Percentage
True Attacks	1,114	0.01%
Attack Attempts	3,948,645	27.37%
Benign Triggers	7,057,012	48.91%
Unknown	3,422,763	23.72%
Total	14,429,534	100%

Table 4.2: **Reason of Alerts**— We categorize different reasons for triggering alerts during the period of 12/2020–07/2022.

Setting	# Alerts	# Alerts w/ Victim Identified	Uniq. # of Victim IPs
Before IP Recovery	14,429,534	7,531,038	18,115
After IP Recovery	49,478,207	37,378,465	69,356

Table 4.3: **Victim IP Recovery**—For network scanning related alerts, the victim IP is not directly available in the alert due to alert aggregation. We recover victim IPs from net connection logs.

4.2.2 Understanding Attack Attempts

In this section, we focus on the 3,948,645 alerts from the second period (2020–2022) that are associated with the attack attempts blocked by BHR, to characterize attacker behaviors.

Geolocation of Attacker IPs. We start by analyzing where the attacker is coming from. For this analysis, we identify *external* attackers that are blocked by BHR. More specifically, the source IP is an external IP (outside of the research center) and the destination IP is an internal IP of the research center. The blocked external IP is then labeled as an *attacker IP* and the internal IP is labeled as the *victim IP*.

In total, we identified 721,169 attacker IPs. By mapping the IPs to their country/region codes (using Maxmind database [77]), we find that the attacker IPs are from all over the world (223 countries/regions). The top five countries are China, the US, India, Brazil, and Germany.

Types of Attacks. Table 4.4 lists the top five alert categories associated with the blocked attack attempts. We observe that the vast majority of the blocked attack attempts are related to network scanning (3 out of 5 categories, 96.7% of the alerts). The other 2 out of 5 are related to brute-force SSH password guessing and suspicious SSH clients.

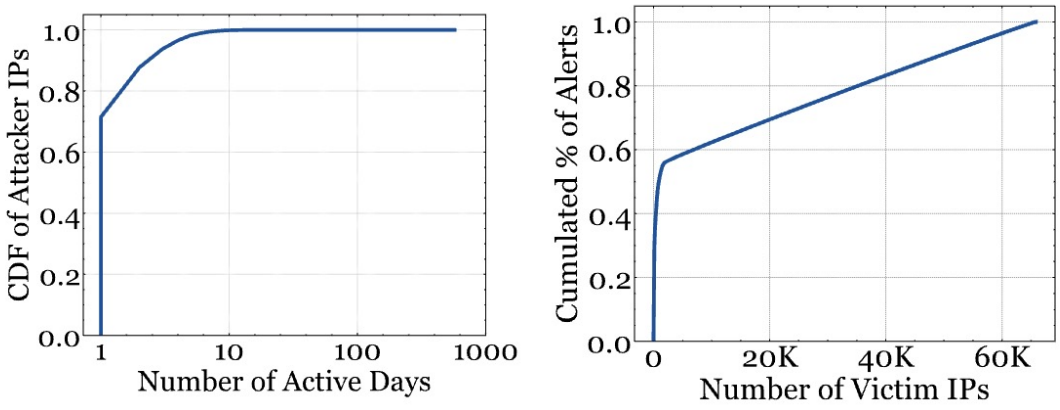

Figure 4.2: **left image**: # of active days per attacker IP. **right image**: Top victim IPs vs. accumulated % of alerts.

Victim IP Recovery. To further analyze attacker behavior, we need to map the attacker-victim pairs based on the alert. Unfortunately, to reduce the volume of *scanning alerts*, Zeek has suppressed and aggregated multiple scanning alerts from the same attacker IP into a single alert (as a summary). For example, if an attacker scanned 200 IPs, a single alert is recorded in the log to summarize the activity *but the list of 200 scanned hosts is omitted* from the alert. This creates difficulties for attributing the victim hosts. As shown in Table 4.3, among the 14.4 million attack-attempt alerts, we can only identify victim IPs for 7.5 million (52%).

To recover the victim IPs, we worked with the Security Operations Center (SOC) to correlate the alert data with network connection logs. The network connections logs are several orders of magnitude larger than the alert log, which can only be stored for two years. As such, we can only perform this recovery analysis for 2020–2022 period. Given an alert, we search the connection log of the same day to identify connections with a matching source IP (attacker IP) and the destination port for each alert. This means a single alert summary will be expanded to multiple alerts (one for each victim host). As a result (see Table 4.3), the number of alerts is increased to 49 million, and the number of victim IPs is increased from 18K to 69K. This recovery method is not necessarily sustainable because the connection logs will be deleted after two years due to its enormous storage requirement. We will use this extended dataset for the following analysis.

Recurrent Attackers. Next, we examine how persistent the attackers are. In other words, once the attacker IP is blocked by BHR (for some period), how likely will they make more attack attempts? Here, for each attacker, we define their *active days* as the number of days when they made attack attempts. Figure 4.2 shows the Cumulative Distribution Function (CDF) for the number of active days per attacker IP. We observe that about 75% of the

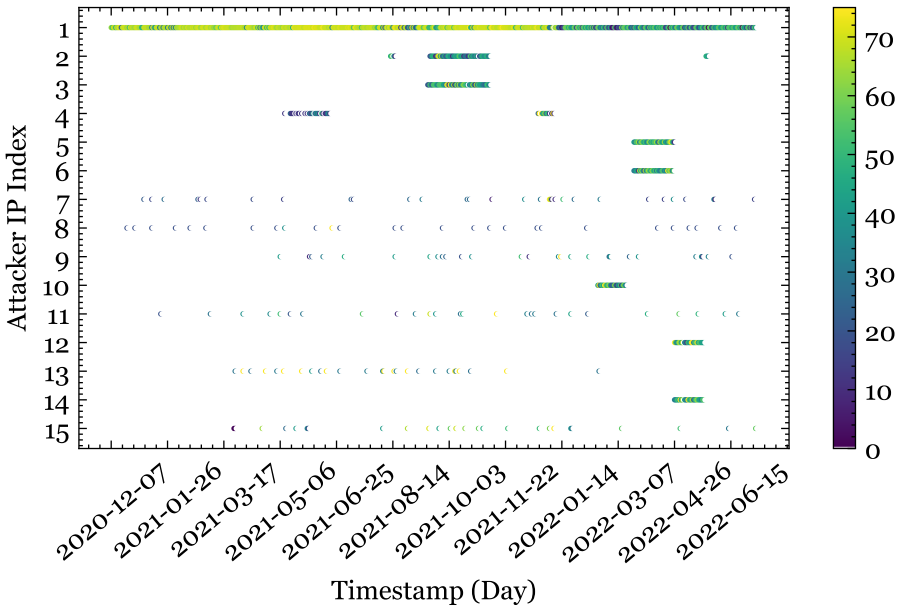

Figure 4.3: Top 15 attackers with the highest number of active days. Each row (y-axis) represents one attacker. The color of the dot represents the number of alerts from this attacker for a given day.

attacker IPs are only active for one day, indicating they are short-lived. However, a small portion of them (0.3%, 28 IPs) have more than a week of active attempts, indicating a high level of persistence.

To understand the activity patterns of top attackers, we plot the heatmap in Figure 4.3 for the top 15 most active attackers (that had at least 26 days of active attack attempts). The top-1 IP is making active attempts for 573 days. After investigation, we discover that this is an (undocumented) external scanner set up by one SOC member to test the BHR system. Other IPs are attacker IPs. Some attackers (e.g., ID=2, 3, 4, 5, 6) had continued with their attack attempts for a concentrated period of time (e.g., several weeks or even months) before stopping. Other attackers (e.g., ID=7, 8, 9) spread their attack attempts across time, and persistently return after certain time gaps.

Uneven Alert Distribution of Victims. Finally, we find that not all victim IPs are evenly targeted. In Figure 4.2, we sort the victim IPs based on the number of associated alerts on the x-axis and report the accumulated percentage of alerts on the y-axis. We observe that a very small portion of victim IPs (1,576, 2.3%) contribute 55% of the total alerts. Then the rest of the IPs (67,780, 97.7%) contributed the rest of 45% of the alerts. This represents an uneven distribution of alerts across hosts. We confirmed with SOC that some of these most-targeted hosts were running services with open access via the public Internet (i.e., more alerts). The skewness is also reflected by the number of alerts per host—

Alert Description	Count (%)
Address Scan	3,368,776 (85.3%)
Random Scan	384,247 (9.7%)
Port Scan	63,676 (1.6%)
SSH PSW Guess	58,879 (1.5%)
Bad SSH Client	40,747 (1.0%)

Table 4.4: **Top 5 Alerts**—Descriptive categories for the top 5 alerts since we cannot present the exact alert name from the SOC.

while the maximum number is 388,502 alerts, the median is only 203 alerts per host (over 2 years). Prior works have proposed per-host models for anomaly detection or alert/event prediction [62, 78, 79, 80]. The skewness of the event distribution can create challenges for such models.

4.3 DATA-SPACE DRIFT IN NETWORK DATA

In this section, we explore the presence of data-space drift within the network dataset obtained from the real-world Security Operations Center (SOC). Our approach involves constructing a network attack detector, which incorporates an innovative feature engineering technique and a Support Vector Machine (SVM)-based model. Through this methodology, we aim to not only ascertain the existence of data-space drift but also enhance the resilience of our detection model in mitigating its impact.

4.3.1 Feature Engineering: Host-level Feature and Cross-host Feature

Host-level Features. The following features are collected and generated for each single connection log extracted from the real-world SOC's network traffic dataset. Generally, we categorize them into two types: continuous features and categorical features.

- Continuous Features:

  Duration: Duration, the temporal aspect of network connections, was scrutinized and binned into seven distinct categories. These categories were intelligently delineated based on statistical analysis, taking into account essential parameters such as mean duration, standard deviation, and outlier detection. By segmenting the duration into discrete intervals, the feature space effectively captured the diverse temporal behaviors exhibited by network connections.

Bytes Transferred: The bytes transferred during network interactions formed a pivotal component of the feature space. Various metrics about data transfer, including original bytes, packets, and IP bytes, were handled as continuous variables. Leveraging binning techniques, these metrics were discretized into meaningful categories, enabling the characterization of diverse data transfer volumes and patterns across the network.

Time Gap: Understanding the temporal dynamics between consecutive network connections was deemed paramount in uncovering latent temporal patterns. The time gap, representing the duration between successive connections, was calculated and subjected to binning procedures. By discretizing time gaps into distinct intervals, the feature space provided profound insights into the temporal dependencies and interconnection intervals prevalent within the network traffic data.

- Categorical Features:

  Protocol (Proto): The categorical nature of network protocols necessitated specialized treatment within the feature engineering pipeline. Protocols such as TCP, UDP, and ICMP were categorized and subjected to one-hot encoding. This transformation facilitated the creation of binary features, effectively capturing the presence or absence of specific protocols within network connections. Such granular representation empowered subsequent analysis to discern protocol-specific behaviors and anomalies.

  Connection State (Conn_state): The state of network connections, formed a critical categorical variable within the feature space. Employing one-hot encoding, distinct connection states were represented as binary features. This encoding schema facilitated comprehensive analysis, allowing for the nuanced exploration of connection statuses and their implications on network behavior. The detail is shown in Table 4.5.

  Service: Network services, representing the diverse array of applications and functionalities accessed during network interactions, were categorized as categorical variables. Through exhaustive analysis of service frequencies and patterns, top services were identified and subjected to one-hot encoding. This transformation enabled the creation of binary features, empowering subsequent analysis to discern service-specific trends and anomalies embedded within the network traffic data.

  History: Historical attributes associated with network connections, including connection history types, constituted a crucial categorical variable within the feature space. Leveraging one-hot encoding, diverse historical contexts were elegantly represented as binary features. This encoding schema facilitated nuanced analysis, enabling the exploration of connection histories and their ramifications on network behavior and

State	Definition
SF	Normal establishment and termination. No byte counts in the summary.
RSTO	Connection established, originator aborted (sent a RST).
REJ	Connection attempt rejected.
S0	Connection attempt seen, no reply.
S1	Connection established, not terminated.
S2	Connection established and close attempt by originator seen (responder no reply).
S3	Connection established and close attempt by responder seen (originator no reply).
RSTR	Responder sent a RST.
RSTOS0	Originator sent an SYN followed by an RST, but no SYN-ACK from the responder.
RSTRH	Responder sent a SYN ACK followed by a RST, no SYN from the originator.
SH	Originator sent an SYN followed by a FIN, but no SYN ACK from the responder.
SHR	Responder sent a SYN ACK followed by a FIN, but no SYN from the originator.
OTH	No SYN seen, just midstream traffic.

Table 4.5: **Definitions**—Network Connection States.

security.

IP Address Binning: IP addresses, serving as fundamental identifiers within network communications, underwent classification into distinct classes. Ranging from Class A (0-126), B (128-191), C (192-233), D (224-239) to Class E (240-254), each IP address class was delineated based on its range and significance within the network topology. This classification schema facilitated in-depth analysis, allowing for the exploration of IP address characteristics and spatial distribution patterns across the network.

Port Binning: Network ports, essential conduits for communication between network entities, were subject to rigorous categorization based on their usage characteristics. By classifying ports into well-known (0-1023), registered (1024-49151), and dynamic/private categories (49152-65535), the feature space effectively captured the diverse behaviors exhibited by network ports. Through binning techniques, port-related features provided profound insights into port usage patterns and their implications on network traffic dynamics.

Cross-host Features. The concept of a host-level feature is valuable in capturing the characteristics and patterns of individual network connection logs. However, its limitation lies in its inability to encapsulate the relationships between different connection logs. This deficiency can result in a decrease in ML model performance since crucial contextual information may be overlooked. To address this, the approach of incorporating cross-host information becomes imperative.

To overcome the information loss associated with host-level features, we propose the development of cross-host features for each connection log. Cross-host features aim to capture

the interactions and dependencies between multiple logs. By doing so, these features enable a more holistic understanding of the network dynamics and facilitate a deeper analysis of the interplay between various connections. We will show the model performance increase and better robustness against data-space drift in Section 4.3.2.

We categorize an IP address as "external" if it falls outside the defined range of internal IP addresses within the Security Operations Center (SOC) network infrastructure. Conversely, an IP address is designated as "internal" if it resides within the SOC's designated internal IP range. These classifications serve as foundational distinctions throughout subsequent sections of this study.

- *Originator Density*: Originator density is a cross-host feature designed to quantify the frequency of external connections emanating from a particular sender within a predefined 24-hour time window, thereby offering a nuanced understanding of the broader network context. To compute this feature, each external originator connection log (if the originator is internal, we simply set this feature to zero) is examined to extract its timestamp, denoting the time at which the connection was established. Subsequently, a retrospective analysis is conducted, spanning the preceding 24-hour period before the timestamp. Within this temporal scope, all connections attributed to the originator are examined, and the count of external connections is tallied. This approach ensures that the metric encapsulates the originator's recent activity, providing insights into potential malicious patterns over time.

- *Responder Density*: Responder density parallels the concept of Originator density, yet focuses on the recipient end of network connections. This feature quantifies the frequency of external connections directed towards a specific receiver within a designated 24-hour timeframe, offering valuable insights into potential threat vectors targeting network endpoints. Unlike Originator density, where the analysis centers on the sender's activity, responder Density evaluates the incoming traffic to a receiver. For each external responder connection log (if the responder is internal, we simply set this feature to zero), the timestamp of the connection establishment is retrieved, enabling an examination of the 24 hours preceding the receiving timestamp. Within this temporal window, all connections destined for the receiver are scrutinized, with the count of external connections tallied.

4.3.2 Modeling and Experiment

In this section, we delve into the modeling process, detailing various experiments conducted to illuminate the presence of data-space drift. Through these experiments, we aim to uncover insights into the impact of different features, discerning the most influential ones that shape both the data distribution alteration and the model performance.

Model and Metric. In our modeling phase, we employ Support Vector Machine (SVM) as our classifier, specifically utilizing the linearSVM variant. The input data format consists of a 1-dimensional vector comprising both host-level and cross-host features, as elaborated in Section 4.3.1. These features are carefully engineered to encapsulate pertinent information regarding network connections. The output of the classifier is a binary value, indicating the classification of the connection log as either malicious or benign, facilitating the detection and identification of potential security threats within the network environment.

In our experiments, we mainly use the following metrics:

- F1 Score: The harmonic mean of precision and recall, F1 Score provides a balanced measure of the model's accuracy, especially in situations where there is an imbalance between the classes. It is calculated as $2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$.

- Area Under the Precision-Recall Curve (AUPRC): AUPRC quantifies the area under the precision-recall curve, depicting the trade-off between precision and recall across various classification thresholds. It offers insights into the model's ability to identify positive instances while minimizing false positives.

- Area Under the Receiver Operating Characteristic Curve (AUC): AUC measures the area under the receiver operating characteristic (ROC) curve, illustrating the classifier's performance across different true positive rates (sensitivity) and false positive rates (1 - specificity). Higher AUC values indicate the superior discrimination ability of the classifier.

- Jensen-Shannon Divergence (JSD): JSD is the symmetric version of the KL divergence. Positive values indicate divergence in distribution. The bigger the value the larger the divergence. In our context, Jensen-Shannon Divergence (JSD) serves as a metric to quantify the dissimilarity between two probability distributions of features. Here, a "feature distribution" refers to a 1-dimensional vector where each entry represents a feature, and the sum of all entries equals 1. These feature distributions are constructed based on statistical observations within specific time windows.

For instance, consider two binary features, A and B, with potential states A1, A2, B1, and B2. Then, a1, a2, b1, b2 are their occurrence within a predefined time window. The construction of the feature distribution involves normalizing the counts of each feature state by the total count of that feature within the given time window. Thus, the feature distribution vector for this example would be a 1x4 vector, with entries calculated as follows:

$$\frac{1}{2}\left[\frac{a1}{a1+a2}, \frac{a2}{a1+a2}, \frac{b1}{b1+b2}, \frac{b2}{b1+b2}\right] \tag{4.1}$$

Measuring Data Distribution Shift. In this section, our focus lies in addressing two pivotal questions: Firstly, does the data distribution shift indeed manifest itself? Secondly, we seek to identify the features that wield significant influence over this shift in data distribution.

In Figure. 4.4, The SVM model is initially trained using data exclusively from day 0. Subsequently, its performance is evaluated through F1 score across the ensuing 190 days. Alongside this evaluation, we compute the Jensen-Shannon Divergence (JSD) between the feature distributions observed on day 0 and each subsequent day n, spanning a range from 1 to 190 days. This comprehensive analysis enables us to gain a nuanced understanding of both the model's predictive capabilities over time and the dynamics of feature distribution shifts.

We observe multiple peaks in both the blue line (representing the Jensen-Shannon Divergence (JSD) between the feature distribution of day 0's malicious logs and subsequent days') and the green line (indicating the JSD between the feature distribution of day 0's benign logs and subsequent days'). These peaks signify substantial disparities in feature distributions over time. Notably, the data distribution shift occurs in both benign and malicious connection logs.

Moreover, the peaks associated with malicious logs exhibit significantly larger values of JSD and are more pronounced. This suggests that the feature distribution of malicious samples undergoes faster and more significant shifts compared to benign samples. Such a phenomenon could be attributed to attackers continually updating their attack techniques or employing highly variable attack patterns to bypass network firewalls.

In Section 4.3.1, we incorporate both host-level and cross-host features into our SVM model. An important question emerges: which features exert a greater influence on the observed data distribution shift? This inquiry holds significance as discerning the relative importance of different features can offer insights into the underlying mechanisms driving

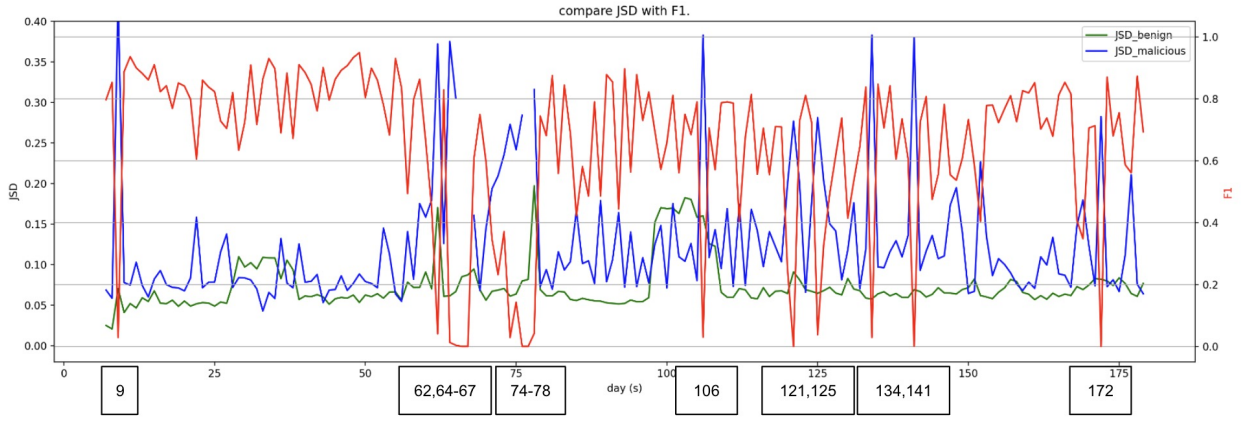

Figure 4.4: Jensen-Shannon Divergence (JSD) and F1 score over 190 days

these shifts, enabling the formulation of more targeted mitigation strategies. To address this question, we experiment. For each feature enumerated in Section 4.3.1, we systematically mask it out by assigning a weight of 0 to it in the final 1-D vector, while redistributing its weight evenly among the remaining features. Subsequently, we recalculate the Jensen-Shannon Divergence (JSD) value between the masked feature distribution and the original distribution from day 0, measuring the change relative to the non-masked JSD value. Significant changes indicate that the masked feature plays a pivotal role in the overall data distribution shift.

In Figure 4.5, we perform experiments in both malicious and benign scenarios to identify the top 5 most influential features contributing to the data distribution shift. Remarkably, a significant overlap exists between these two sets of features. Notably, conn_state, service, history, and cross_log features emerge as common contributors in both cases.

Data-Space Drift. In Section 4.3.2, we conducted several experiments to demonstrate the presence of data distribution shifts in network data spanning a considerable time frame (190 days). Notably, our findings indicate that malicious network data exhibits a more pronounced shift compared to benign data. Furthermore, through a series of mask-out experiments, we identified conn_state, service, history, and cross_log features as significant contributors to this observed data distribution shift.

In this section, we aim to address the following questions:

Firstly, is there an internal relationship between data distribution shift and model performance? Does data-space drift manifest in network traffic data? Secondly, does the incorporation of cross-host features enhance the robustness and reliability of our SVM model against potential data-space drift?

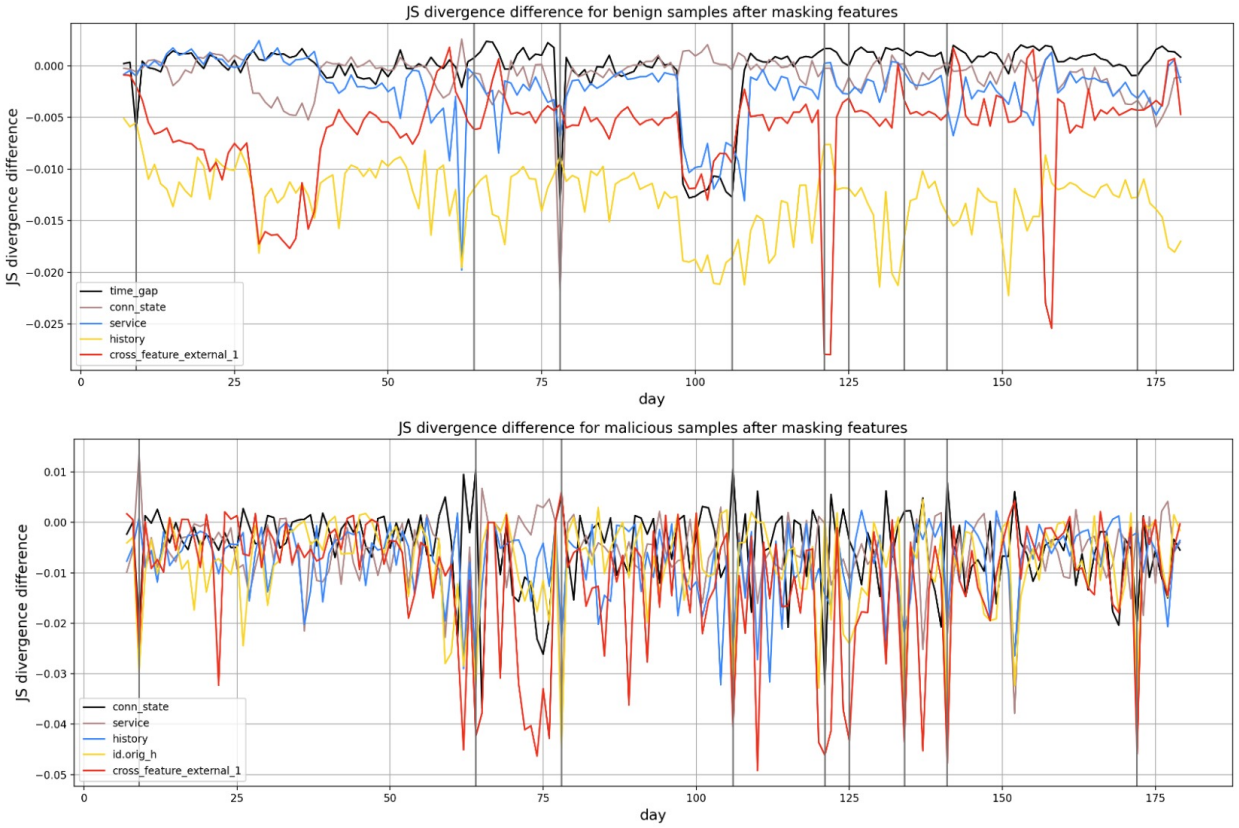

Figure 4.5: **Top features contributing to data distribution shift**—We list the top 5 features in malicious/benign cases that are important to data distribution shift.

In Figure 4.4, a notable pattern emerges where dips in the F1 score align with peaks in the JSD values. Specifically, instances of this alignment occur on day 9, day 62, day 64-67, day 74-78, day 106, day 121, day 125, day 134, day 141, and day 172. The peak in JSD values signifies a substantial disparity between the feature distribution of a particular day and that of day 0, while the dip in the F1 score indicates a decline in model performance.

This observation supports the existence of data-space drift, evidenced by the recurrent dips in the F1 score line. Furthermore, it suggests an internal relationship between data distribution shift and model performance. This relationship can be explained by the fact that when the underlying data distribution shifts significantly, the model trained on the initial distribution may struggle to generalize effectively to the evolving data landscape. Consequently, performance metrics such as the F1 score may suffer as the model's ability to accurately classify instances diminishes in the face of data distribution shifts.

This leads to a natural question: which features contribute significantly to the observed data-space drift, leading to a decrease in model performance? Moreover, do these features

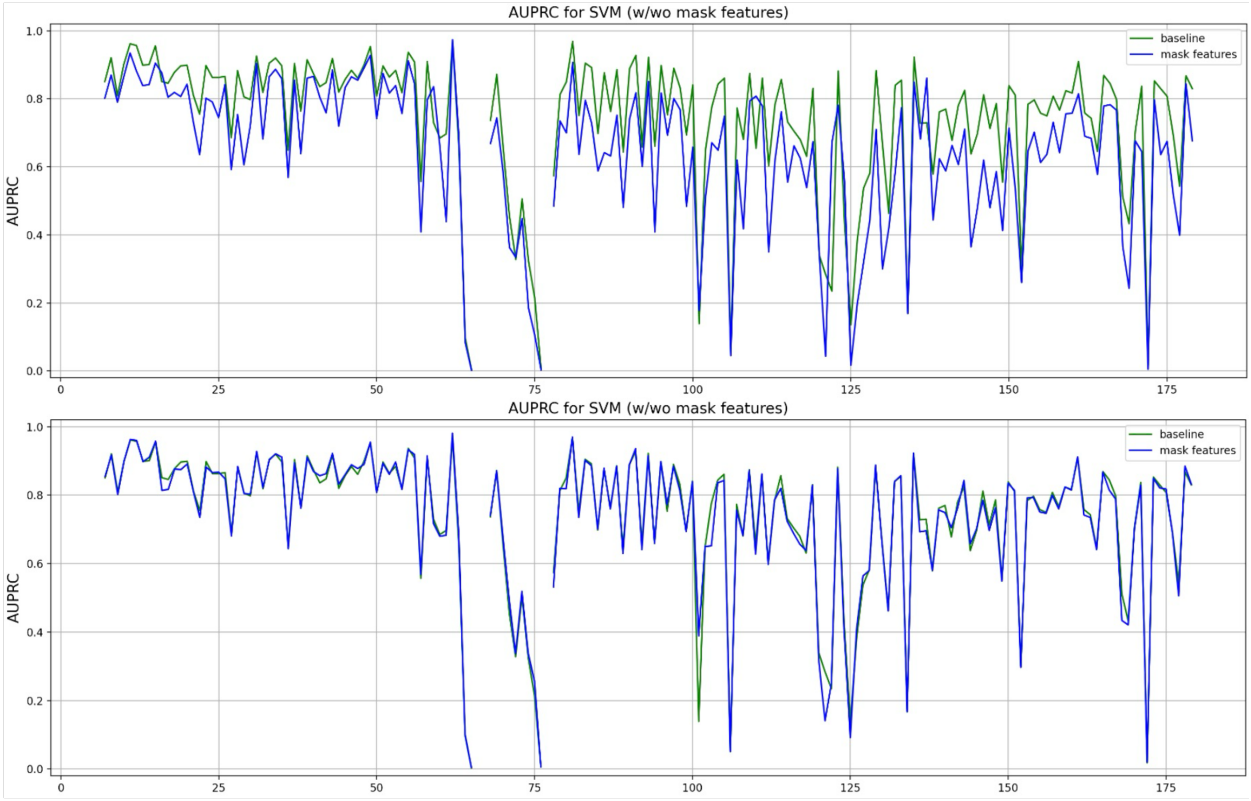

Figure 4.6: **Feature ablation performance experiment**—Above is the AUPRC curve across 190 days with/without masking out cross-host feature, while below is the AUPRC curve with/without masking out the conn_state feature.

align with those contributing to the data distribution shift, as explored in Section 4.3.2? To address this, we conduct mask-out experiments once more. Specifically, we mask out conn_state and cross_log features separately to observe how the model performance changes. Comparing these results will help us discern the specific features that play a pivotal role in both the data distribution shift and the subsequent decline in model performance.

In Figure 4.6, two intriguing observations emerge. Firstly, the blue lines consistently lie below the green lines. This indicates that masking out conn_state or cross-host features individually results in a decline in model performance. Given our findings from Section 4.3.2, where we identified conn_state and cross-host features as contributors to significant data distribution shifts.

Secondly, despite observing a lighter data distribution shift when cross-host features are masked out, the model's performance still decreases. This highlights that while smaller data distribution shifts may occur, they do not guarantee immunity from data-space drift. The persistence of performance decline suggests that cross-host features play a crucial role

in maintaining model reliability even in the presence of milder data distribution shifts. Data-space drift exists because we observe many dips in almost all figures related to model performance, but Figure 4.6 clearly shows the existence of cross-host feature can increase the model performance, leading to better robustness.

CHAPTER 5: DISCUSSION AND CONCLUSION

Within this thesis, our primary focus lies in measuring concept drift within malware and network intrusion detection models, an issue increasingly menacing to cybersecurity and model robustness. Concept drift, manifested through shifts in ML model decision boundaries due to alterations in underlying data distributions or feature spaces, poses significant challenges.

Our exploration comprises two technical experiments aimed at dissecting the causes and consequences of concept drift. In the initial experiment, we dissect the distinct impacts of feature-space and data-space drift on malware detection systems' effectiveness. Our empirical inquiry reveals a discernible trend: while data-space drift predominantly undermines model performance over time, adjustments in feature space seem inconsequential, if not negligible. This trend holds across various feature types and engineering methodologies, encompassing malware detectors tailored for both Android and Portable Executable (PE) files.

These findings carry substantial implications. They imply that a robust, well-constructed feature set can support model re-training efforts to counter data-space drift without necessitating feature set modifications, even amidst an influx of new features introduced by newly analyzed samples. Nonetheless, it would be premature to dismiss the importance of updating features entirely. The nuanced interplay between feature-space stability and model adaptability warrants further investigation.

On the opposing side of the discourse, refraining from frequent feature set updates offers tangible benefits. It allows for greater flexibility in model selection and methodologies employed for incremental updates during re-training sessions. Adopting a novel feature set with each re-training phase entails either a comprehensive model overhaul—a time-consuming, resource-intensive process—or resorting to simpler, potentially less effective linear models for incremental updates. Moreover, frequent feature set updates may inadvertently introduce short-term beneficial features that compromise long-term model accuracy and reliability.

These considerations underscore the necessity for a deeper comprehension of the implications of feature-space updating. Further research in this domain is imperative to devise strategies optimizing the balance between feature stability and model responsiveness to evolving malware landscapes.

In the second experiment, we analyze a real-world network dataset from an anonymous Security Operations Center (SOC) to elucidate network attack characteristics and behaviors. Additionally, we introduce a novel feature engineering method to generate both host-level

and cross-host features, employing an SVM model as a network intrusion detector. Subsequently, we investigate data distribution shift, scrutinizing its impact on model performance. Our analysis reveals several intriguing insights: the existence of data space drift, the multifaceted nature of factors influencing model performance, and the efficacy of cross-host features in enhancing model robustness against data space drift.

While our findings shed light on the potential of cross-host features to enhance model performance amidst concept drift, the relationship between model performance and data distribution shift remains incompletely understood. Despite the promising impact of cross-host features, the persistence of performance dips—indicative of rapid drops in performance—suggests that challenges persist. Moving forward, further exploration is essential to address these lingering issues and devise effective solutions.

REFERENCES

[1] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens, "Drebin: Effective and explainable detection of android malware in your pocket." in *Proc. of NDSS*, 2014.

[2] M. Lindorfer, M. Neugschwandtner, and C. Platzer, "Marvin: Efficient and comprehensive mobile app classification through static and dynamic analysis," in *Prof. of COMPSAC*, 2015.

[3] H. S. Anderson and P. Roth, "Ember: an open dataset for training static pe malware machine learning models," *arXiv preprint arXiv:1804.04637*, 2018.

[4] Y. Chen, S. Wang, D. She, and S. Jana, "On training robust PDF malware classifiers," in *Proc. of USENIX Security*, 2020.

[5] D. Arp, E. Quiring, F. Pendlebury, A. Warnecke, F. Pierazzi, C. Wressnegger, L. Cavallaro, and K. Rieck, "Dos and don'ts of machine learning in computer security," in *Proc. of USENIX Security*, 2022.

[6] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM computing surveys (CSUR)*, 2014.

[7] L. Yang, W. Guo, Q. Hao, A. Ciptadi, A. Ahmadzadeh, X. Xing, and G. Wang, "CADE: Detecting and explaining concept drift samples for security applications," in *Proc. of USENIX Security*, 2021.

[8] D. Han, Z. Wang, W. Chen, Y. Zhong, S. Wang, H. Zhang, J. Yang, X. Shi, and X. Yin, "Deepaid: Interpreting and improving deep learning-based anomaly detection in security applications," in *Proc. of CCS*, 2021.

[9] R. Jordaney, K. Sharad, S. K. Dash, Z. Wang, D. Papini, I. Nouretdinov, and L. Cavallaro, "Transcend: Detecting concept drift in malware classification models," in *Proc. of USENIX Security*, 2017.

[10] F. Barbero, F. Pendlebury, F. Pierazzi, and L. Cavallaro, "Transcending transcend: Revisiting malware classification in the presence of concept drift," in *Proc. of IEEE S&P*, 2022.

[11] K. Xu, Y. Li, R. Deng, K. Chen, and J. Xu, "Droidevolver: Self-evolving android malware detection system," in *Proc. of Euro S&P*, 2019.

[12] Z. Kan, F. Pendlebury, F. Pierazzi, and L. Cavallaro, "Investigating labelless drift adaptation for malware detection," in *Proc. of AISec*, 2021.

[13] A. Narayanan, M. Chandramohan, L. Chen, and Y. Liu, "Context-aware, adaptive, and scalable android malware detection through online learning," *tetci*, 2017.

[14] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in *Proc. of IEEE S&P*, 2010.

[15] M. Baena-Garcıa, J. del Campo-Ávila, R. Fidalgo, A. Bifet, R. Gavalda, and R. Morales-Bueno, "Early drift detection method," in *Proc. of Workshop on knowledge discovery from data streams*, 2006.

[16] A. Bifet and R. Gavalda, "Learning from time-changing data with adaptive windowing," in *Proc. of SDM*, 2007.

[17] M. Harel, S. Mannor, R. El-Yaniv, and K. Crammer, "Concept drift detection through resampling," in *Proc. of ICML*, 2014.

[18] D. M. dos Reis, P. Flach, S. Matwin, and G. Batista, "Fast unsupervised online drift detection using incremental kolmogorov-smirnov test," in *Proc. of KDD*, 2016.

[19] M. Ahmadi, D. Ulyanov, S. Semenov, M. Trofimov, and G. Giacinto, "Novel feature extraction, selection and fusion for effective malware family classification," in *Proc. of CODASPY*, 2016.

[20] T. Chakraborty, F. Pierazzi, and V. Subrahmanian, "Ec2: Ensemble clustering and classification for predicting android malware families," *IEEE TDSC*, 2017.

[21] A. Kantchelian, S. Afroz, L. Huang, A. C. Islam, B. Miller, M. C. Tschantz, R. Greenstadt, A. D. Joseph, and J. D. Tygar, "Approaches to adversarial drift," in *Proc. of AISec*, 2013.

[22] F. Pendlebury, F. Pierazzi, R. Jordaney, J. Kinder, and L. Cavallaro, "TESSERACT: Eliminating experimental bias in malware classification across space and time," in *Proc. of USENIX Security*, 2019.

[23] G. Andresini, F. Pendlebury, F. Pierazzi, C. Loglisci, A. Appice, and L. Cavallaro, "Insomnia: Towards concept-drift robustness in network intrusion detection," in *Proc. of AISec*, 2021.

[24] X. Zhang, Y. Zhang, M. Zhong, D. Ding, Y. Cao, Y. Zhang, M. Zhang, and M. Yang, "Enhancing state-of-the-art classifiers with api semantics to detect evolved android malware," in *Proc. of CCS*, 2020.

[25] F. B. Kokulu, A. Soneji, T. Bao, Y. Shoshitaishvili, Z. Zhao, A. Doupé, and G.-J. Ahn, "Matched and mismatched socs: A qualitative study on security operations center issues," in *Proc. of CCS*, 2019.

[26] B. A. Alahmadi, L. Axon, and I. Martinovic, "99% false positives: A qualitative study of SOC analysts' perspectives on security alarms," in *Proc. of USENIX Security*, 2022.

[27] C. Zimmerman, *Ten Strategies of a World-Class Cybersecurity Operations Center*. MITRE Corporation, 2014.

[28] R. Stevens, D. Votipka, J. Dykstra, F. Tomlinson, E. Quartararo, C. Ahern, and M. L. Mazurek, "How ready is your ready? assessing the usability of incident response playbook frameworks," in *Proc. of CHI*, 2022.

[29] D. J. Chaboya, R. A. Raines, R. O. Baldwin, and B. E. Mullins, "Network intrusion detection: Automated and manual methods prone to attack and evasion," *IEEE Security and Privacy*, 2006.

[30] T.-F. Yen, A. Oprea, K. Onarlioglu, T. Leetham, W. Robertson, A. Juels, and E. Kirda, "Beehive: Large-scale log analysis for detecting suspicious activity in enterprise networks," in *Proc. of ACSAC*, 2013.

[31] S. Oesch, R. Bridges, J. Smith, J. Beaver, J. Goodall, K. Huffer, C. Miles, and D. Scofield, "An assessment of the usability of machine learning based tools for the security operations center," in *Proc. of iThings*, 2020.

[32] V. G. Li, M. Dunn, P. Pearce, D. McCoy, G. M. Voelker, S. Savage, and K. Levchenko, "Reading the tea leaves: A comparative analysis of threat intelligence," in *Proc. of USENIX Security*, 2019.

[33] V. Paxson, "Bro: A system for detecting network intruders in Real-Time," in *Proc. of USENIX Security*, 1998.

[34] J. Goodall, W. Lutters, and A. Komlodi, "I know my network: Collaboration and expertise in intrusion detection," in *Proc. of CSCW*, 2004.

[35] O. Akinrolabu, I. Agrafiotis, and A. Erola, "The challenge of detecting sophisticated attacks: Insights from soc analysts," in *Proc. of ARES*, 2018.

[36] E. Agyepong, Y. Cherdantseva, P. Reinecke, and P. Burnap, "Challenges and performance metrics for security operations center analysts: a systematic review," *Journal of Cyber Security Technology*, 2020.

[37] S. C. Sundaramurthy, A. G. Bardas, J. Case, X. Ou, M. Wesch, J. McHugh, and S. R. Rajagopalan, "A human capital model for mitigating security analyst burnout," in *Proc. of SOUPS*, 2015.

[38] S. C. Sundaramurthy, J. McHugh, X. Ou, M. Wesch, A. G. Bardas, and S. R. Rajagopalan, "Turning contradictions into innovations or: How we learned to stop whining and improve security operations," in *Proc. of SOUPS*, 2016.

[39] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in *Proc. of IEEE S&P*, 2010.

[40] G. Vigna and R. A. Kemmerer, "Netstat: A network-based intrusion detection system," *Journal of computer security*, 1999.

[41] A. Javaid, Q. Niyaz, W. Sun, and M. Alam, "A deep learning approach for network intrusion detection system," in *Proc. of BICT*, 2016.

[42] A. Pecchia, A. Sharma, Z. Kalbarczyk, D. Cotroneo, and R. K. Iyer, "Identifying compromised users in shared computing infrastructures: A data-driven bayesian network approach," in *Proc. of SRDS*, 2011.

[43] X. Han, T. F. J. Pasquier, A. Bates, J. Mickens, and M. I. Seltzer, "Unicorn: Runtime provenance-based detector for advanced persistent threats," in *Proc. of NDSS*, 2020.

[44] W. U. Hassan, A. Bates, and D. Marino, "Tactical provenance analysis for endpoint detection and response systems," in *Proc. of IEEE S&P*, 2020.

[45] M. Liu, Z. Xue, X. Xu, C. Zhong, and J. Chen, "Host-based intrusion detection system with system calls: Review and future trends," *ACM Computing Surveys (CSUR)*, 2018.

[46] Y.-j. Ou, Y. Lin, and Y. Zhang, "The design and implementation of host-based intrusion detection system," in *Proc. of IITSI*, 2010.

[47] B. Bowman, C. Laprade, Y. Ji, and H. H. Huang, "Detecting lateral movement in enterprise computer networks with unsupervised graph AI," in *Proc. of RAID*, 2020.

[48] F. Dong, L. Wang, X. Nie, F. Shao, H. Wang, D. Li, X. Luo, and X. Xiao, "Distdet: A cost-effective distributed cyber threat detection system," in *Proc. of USENIX Security*, 2023.

[49] A. Årnes, F. Valeur, G. Vigna, and R. A. Kemmerer, "Using hidden markov models to evaluate the risks of intrusions," in *Proc. of RAID*, D. Zamboni and C. Kruegel, Eds., 2006.

[50] W. U. Hassan, S. Guo, D. Li, Z. Chen, K. Jee, Z. Li, and A. Bates, "Nodoze: Combatting threat alert fatigue with automated provenance triage," *Proc. of NDSS*, 2019.

[51] A. AlEroud and G. Karabatis, "Beyond data: Contextual information fusion for cyber security analytics," in *Proc. of SAC*, 2016.

[52] T. Ban, N. Samuel, T. Takahashi, and D. Inoue, "Combat security alert fatigue with ai-assisted techniques," in *Proc. of CSET Workshop*, 2021.

[53] C. Krügel and W. K. Robertson, "Alert verification determining the success of intrusion attempts," in *Proc. of DIMVA*, 2004.

[54] S. Salah, G. Maciá-Fernández, and J. E. Díaz-Verdejo, "A model-based survey of alert correlation techniques," *Computer Networks*, 2013.

[55] T. van Ede, H. Aghakhani, N. Spahn, R. Bortolameotti, M. Cova, A. Continella, M. van Steen, A. Peter, C. Kruegel, and G. Vigna, "DeepCASE: Semi-Supervised Contextual Analysis of Security Events," in *Proc. of IEEE S&P*, 2022.

[56] E. Raftopoulos, M. Egli, and X. Dimitropoulos, "Shedding light on log correlation in network forensics analysis," in *Proc. of DIMVA*, U. Flegel, E. Markatos, and W. Robertson, Eds., 2013.

[57] A. Sharma, Z. Kalbarczyk, J. Barlow, and R. Iyer, "Analysis of security data from a large computing organization," in *Proc. of DSN*, 2011.

[58] G. Ho, A. Sharma, M. Javed, V. Paxson, and D. Wagner, "Detecting credential spearphishing in enterprise settings," in *Proc. of USENIX Security*, 2017.

[59] J. Lee, F. Tang, P. M. Thet, D. Yeoh, M. Rybczynski, and D. M. Divakaran, "SIERRA: ranking anomalous activities in enterprise networks," in *Proc. of EuroS&P*, 2022.

[60] A. Alsaheel, Y. Nan, S. Ma, L. Yu, G. Walkup, Z. B. Celik, X. Zhang, and D. Xu, "ATLAS: A sequence-based learning approach for attack investigation," in *Proc. of USENIX Security*, 2021.

[61] S. M. Milajerdi, B. Eshete, R. Gjomemo, and V. Venkatakrishnan, "Poirot: Aligning attack behavior with kernel audit records for cyber threat hunting," in *Proc. of CCS*, 2019.

[62] Y. Shen, E. Mariconti, P. A. Vervier, and G. Stringhini, "Tiresias: Predicting security events through deep learning," in *Proc. of CCS*, 2018.

[63] L. Allodi and F. Massacci, "Security events and vulnerability data for cybersecurity risk estimation," *Risk Analysis*, 2017.

[64] Z. Chen, Z. Zhang, Z. Kan, L. Yang, J. Cortellazzi, F. Pendlebury, F. Pierazzi, L. Cavallaro, and G. Wang, "Is it overkill? analyzing feature-space concept drift in malware detectors," in *2023 IEEE Security and Privacy Workshops (SPW)*, 2023.

[65] K. Allix, T. F. Bissyandé, J. Klein, and Y. Le Traon, "Androzoo: Collecting millions of android apps for the research community," in *Proc. of MSR*, 2016.

[66] L. Yang, Z. Chen, J. Cortellazzi, F. Pendlebury, K. Tu, F. Pierazzi, L. Cavallaro, and G. Wang, "Jigsaw puzzle: Selective backdoor attack to subvert malware classifiers," *arXiv preprint arXiv:2202.05470*, 2022.

[67] F. Pierazzi, F. Pendlebury, J. Cortellazzi, and L. Cavallaro, "Intriguing properties of adversarial ML attacks in the problem space," in *Proc. of IEEE S&P*, 2020.

[68] A. Demontis, M. Melis, B. Biggio, D. Maiorca, D. Arp, K. Rieck, I. Corona, G. Giacinto, and F. Roli, "Yes, machine learning can be more secure! a case study on android malware detection," *IEEE TDSC*, 2017.

[69] K. Xu, Y. Li, R. H. Deng, and K. Chen, "Deeprefiner: Multi-layer android malware detection system applying deep neural networks," in *Proc. of Euro S&P*, 2018.

[70] G. Severi, J. Meyer, S. Coull, and A. Oprea, "Explanation-guided backdoor poisoning attacks against malware classifiers," in *Proc. of USENIX Security*, 2021.

[71] H. Li, S. Zhou, W. Yuan, X. Luo, C. Gao, and S. Chen, "Robust android malware detection against adversarial example attacks," in *Proc. of WWW*, 2021.

[72] W. Wu, B. Li, L. Chen, J. Gao, and C. Zhang, "A review for weighted minhash algorithms," *IEEE TKDE*, vol. 34, no. 6, pp. 2553–2573, 2022.

[73] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," in *Proc. of NeurIPS*, 2017.

[74] L. Li, "Splunk dashboard studio: Dashboard customization," https://www.splunk.com/en_us/blog/platform/dashboard-studio-dashboard-customization-made-easy.html, 2021.

[75] Zeek, "The zeek network security monitor," https://zeek.org/, 2023.

[76] W. Kumari and D. McPherson, "RFC5635: Remote triggered black hole filtering with unicast reverse path forwarding (urpf)," https://www.rfc-editor.org/rfc/rfc5635, 2009.

[77] "Maxmind," https://www.maxmind.com/, 2023.

[78] H. Bian, T. Bai, M. A. Salahuddin, N. Limam, A. Abou Daya, and R. Boutaba, "Host in danger? detecting network intrusions from authentication logs," in *CNSM*, 2019.

[79] M. S. Ansari, V. Bartoš, and B. Lee, "Gru-based deep learning approach for network intrusion alert prediction," *Future Generation Computer Systems*, vol. 128, pp. 235–247, 2022.

[80] M. Husák, J. Komárková, E. Bou-Harb, and P. Čeleda, "Survey of attack projection, prediction, and forecasting in cyber security," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 1, pp. 640–660, 2019.