Model for Detection of Masquerade Attacks Based on Variable-Length Sequences

GHAZAROS BARSEGHYAN, YUYU YUAN, AND MANAWA ANAKPA
School of Computer Science (National Pilot Software Engineering School), Beijing University of Posts and Telecommunications, Beijing 100876, China
Key Laboratory of Trustworthy Distributed Computing and Service, Ministry of Education, Beijing 100876, China
Corresponding author: Ghazaros Barseghyan (kazaros91@gmail.com)
This work was supported by the National Natural Science Foundation of China under Grant 91118002.

ABSTRACT A masquerader is an attacker who gains illegitimate access to a user’s account. Masquerade detection is one of the key problems of intrusion detection systems. Deep learning models that obtained state-of-the-art results in masquerade detection have failed to exhibit very high detection performance when data samples contain limited information. Alternatively, computationally cheaper and more memory-efficient traditional machine learning models suffer from less robust features, which hinders them in achieving high detection performance. The contributions of this article are as follows: we introduce new features of variable-length UNIX command sequences (i.e., weighted occurrence frequencies of different orders) and integrate these features into an extended Markov-chain-based variable-length model. The detection performance of our model is evaluated on three publicly available and free datasets: Schonlau (SEA), Purdue (PU), and Greenberg. The results demonstrate that our model significantly improves the true positive rate (TPR), false positive rate, receiver operator characteristic, and threshold variance compared to the baselines (other Markov-chain-based variable-length models). Furthermore, in terms of the TPR, the proposed method is superior to a state-of-the-art deep learning model that uses a convolutional neural network on the PU and Greenberg datasets and a state-of-the-art sequence-alignment-hidden Markov model on the SEA dataset. Moreover, the proposed method is much more lightweight than the state-of-the-art models in terms of computational and memory complexity, and thus more suitable for real-time masquerade detection.

INDEX TERMS Anomaly detection, intrusion detection, machine learning (ML), Markov chain, masquerade detection, UNIX commands.

I. INTRODUCTION
The Internet is an indispensable part of people’s lives and has given rise to the rapid growth of e-commerce and online social networks. Owing to the increasing number of users, problems with user trustworthiness and security are becoming increasingly important. An intrusion detection system (IDS) is a software-based application or hardware device that is used to identify malicious behavior in a network [1], [2]. The most common types of attack include masquerading, malware, spyware, denial of service, probe, user-to-root (U2R), and remote-to-user (R2L). In this study, we focused on the problem of masquerade detection. A masquerader is an attacker who gains illegitimate access to a user’s account. In other words, masqueraders impersonate legitimate users [3].

For example, masqueraders can be insiders of a system who can cause severe harm, such as stealing valuable confidential information or disrupting data integrity. Therefore, masquerade detection is very important for the security and trustworthiness of a system.

To build an IDS, some authors applied traditional machine learning (ML) models, while others used deep learning (DL) models. Such models are used in IDS for misuse detection, anomaly detection, or a combination of these approaches. For misuse detection, assumptions are made about the attacker’s strategy; training data need to be labeled; hence supervised learning methods are employed. The main disadvantage of the misuse detection approach is the difficulty of adapting to dynamic changes in attacking behavior. If an attacker’s strategy changes, then the detection approach needs to be updated [4]–[6]. In this study, therefore, we opted for the anomaly detection approach. Anomaly detection is focused
on finding questionable behavior patterns that cannot match those expected from data [7]. In the training stage, features are extracted to model normal user behavior. In the detection stage, an audited user’s behavior is classified as normal or anomalous (or doubtful as done by Dash et al. [8]) based on its deviation from the modeled normal behavior.

Many authors have proposed various methods based on anomaly detection for masquerade detection using UNIX commands. Most methods use three publicly available and free UNIX command-line-based datasets: Schonlau (SEA) [9], Purdue (PU) [10], and Greenberg [11]. Two data configurations are used in the literature: truncated and enriched. The truncated data configuration indicates that a dataset contains only command names. In contrast, the enriched data configuration indicates that a dataset consists of command names, parameters, flags, shell metacharacters, and information about the start and end of user sessions.

Schonlau et al. [9] released the SEA dataset with only the truncated data configuration. In contrast, the PU and Greenberg datasets were released with the enriched data configurations. In the literature, PU and Greenberg datasets were used not only with the enriched configuration, but also with the truncated configuration. We name the datasets with the respective data configurations as PU Enriched, PU Truncated, Greenberg Enriched, and Greenberg Truncated.

Most of the related papers used the data settings that yield a normal test set and masquerade test set, as proposed by Schonlau et al. [9] for SEA (SEA Full), Maxion [12] for Greenberg, and Lane and Brodley [10] for PU. We named these data settings as “Full.” With Full data settings, PU Truncated, PU Enriched, Greenberg Truncated, and Greenberg Enriched are denoted by PU Truncated Full, PU Enriched Full, Greenberg Truncated Full, and Greenberg Enriched Full, respectively. We describe these datasets with the associated data settings in detail in Section III-A.

The previous work exhibited relatively high detection performance on the datasets with the enriched data configuration but suffered from significantly worse results with the truncated data configuration. DL models that obtained state-of-the-art results in masquerade detection have failed to exhibit very high detection performance when data samples contain limited information. Alternatively, traditional ML models that are more computationally- and memory-efficient, suffer from less robust features, which hinders them in achieving higher detection performance. Therefore, in this study, we focused on experimenting on the datasets with the truncated data configuration only, that is, when the data samples contain limited information.

A. CONTRIBUTIONS
In this article, we make the following five contributions:

- We extend the Markov chain-based variable-length model of Xiao et al. [4] and integrate the new features into the model (see Sections II-B.5–II-B.7).
- On three different datasets (SEA, PU Truncated, and Greenberg Truncated) with two data settings, our model outperforms the baselines (Xiao et al. [4]) at several metrics, such as the true positive rate (TPR), false positive rate (FPR), receiver operator characteristic (ROC), and threshold variance.
- Our model achieved significant improvement on the TPR metric compared to state-of-the-art Convolutional Neural Network CNN [13] for PU Truncated Full, Greenberg Truncated Full, and state-of-the-art sequence-alignment Hidden Markov Model SA-HMM [14] for SEA Full.
- Our model requires much less computation and memory usage than the state-of-the-art models and is therefore more suitable for masquerade detection in real-time.

B. RELATED WORK

1) MASQUERADE DETECTION BASED ON ANOMALY DETECTION
Maxion [12] applied a Naïve Bayes classifier for masquerade detection and tested it on SEA Full with two variations: with updating user profiles (TPR = 61.5%, FPR = 1.3%) and without updating them (TPR = 66.2%, FPR = 4.6%). Maxion also introduced new data setting (SEA1v49) for which the first model realized a TPR of 62.8% and FPR of 4.6%. This model obtained a TPR of 70.9% and FPR of 4.7% for Greenberg Truncated Full, and a TPR of 82.1% and FPR of 5.7% for Greenberg Enriched Full. Yung [15] extended the work of Maxion [12] by employing the Naïve Bayes classifier with both updating and feedback. For SEA Full, his model obtained a TPR of 76% and FPR of 2%. Improving upon his own work, Yung [16] employed a self-consistent Naïve Bayes model with updating and obtained a TPR of 79%, although the FPR was still 2%. In contrast, Sharma and Paliwal explored a Naïve Bayes approach combined with weighted radial basis similarity measure (WRBF-NB) [17]. For SEA Full, WRBF-NB obtained a high TPR of 83.1%, although FPR of 7.7% was also high. In [18], Reddy et al. applied a conditional Naïve Bayes classifier and achieved a TPR of 84% and FPR of 8.8% for SEA Full. Besides, the model obtained a TPR of 71.1% and FPR of 6% for Greenberg Truncated Full and a TPR of 84.13% and FPR of 9.4% for Greenberg Enriched Full. Furthermore, the model demonstrated a TPR of 84% and FPR of 8% for PU Enriched Full.

Oka et al. [19] used the Eigen co-occurrence matrix (ECM) to model normal user behavior and performed masquerade detection based on the inner product between the ECMs of a normal user and the monitored user. For SEA Full, their method obtained a TPR of 72.3% and FPR of 2.5%. The main drawback of their approach is that calculating ECM is computationally expensive and uses large amounts of memory. By contrast, Kim and Cha [3] proposed a more
lightweight approach by introducing the notion of common commands as a feature and trained a support vector machine (SVM) with a voting engine as a classifier for masquerade detection. They achieved a TPR of 80.1% and FPR of 9.7% for SEA Full, and a TPR of 94.8% and FPR of 0% for SEA1v49. Furthermore, for Greenberg Enriched Full, they obtained a TPR of 87.3% and FPR of 6.4%, whereas for Greenberg Truncated Full they obtained a TPR of 71.1% and FPR of 6%. To obtain more balanced accuracy than that of Kim and Cha [3], and to reduce the calculation burden of Oka et al.’s [19] ECM, Chen and Aritsugi [20] combined ECM with SVM using online updating. They tested their model for one-class (TPR = 72.24%, FPR = 3%) classification for SEA Full. Furthermore, Li et al. [21] first extracted the principal features of the user behavior from a correlation Eigen matrix by principal component analysis (PCA) and then fed these features into an SVM-based detection system. For SEA Full, they achieved the best TPR of 82.6% and FPR of 3% among SVM-based models reported in the literature. Nonetheless, one of the disadvantages of SVM-based models is that they did not demonstrate very good results for SEA Full and Greenberg Truncated Full. Their performance for PU Truncated Full has not been reported in the literature.

Lane and Brodley [22] used two approaches for masquerade detection: Hidden Markov Model (HMM) and instance-based learning. Their HMM-based method obtained a TPR of 62% and FPR of 1.5% for SEA Full. By contrast, Okamoto et al. [23] proposed an immunity-based HMM and obtained a TPR of 60% and FPR of 1% for SEA Full. To enhance the detection performance of an HMM-based approach, Huang and Stamp [24] exploited the positional information of users and proposed a profile HMM (PHMM) that achieved a TPR of 70% and FPR of 5% for SEA Full. Kholidy et al. [25] proposed the data-driven semi-global alignment (DDSGA) approach. In the training phase, a given user’s sequence alignment parameters were calculated. In the detection phase, discovering several misalignment sequences indicated masquerade activities. DDSGA achieved a TPR of 88.4% and FPR of 1.7% for SEA Full. To further improve the detection performance, Qiu et al. [14] combined a sequence alignment (SA) module with an HMM and named it SA-HMM. The SA module mitigated the problem of variations in user activity sequences, and the HMM leveraged the positional information between the observations of users. For SEA Full, SA-HMM achieved a state-of-the-art TPR of 94.1%. However, there is still room for improvement. Furthermore, it is worth mentioning that HMM-based models are computationally costly.

In [9], Schonlau et al. applied a Bayes one-step Markov chain approach and obtained a TPR of 69.3% and FPR of 6.7% in detecting masquerade attacks for SEA Full. In the training stage, transition probabilities were calculated from one command to the next. In the detection stage, they checked whether the observed transition probabilities were normal or anomalous. By contrast, Ju and Vardi [26] explored a hybrid multistep Markov chain for profiling normal transition probabilities between commands. They used maximum likelihood estimation to detect masquerade activities obtaining a TPR of 49.3% and FPR of 3.2% for SEA Full. Nonetheless, the TPR results of the methods used in [9] and [26] are low. Tian et al. [27] used variable-length UNIX command sequences to mine user behavioral patterns and employed similarity measure as a classifier examining each command and each sequence of commands. In [28], the authors continued the work of Tian et al. [27], and considered the transition relations between adjacent commands to develop an approach using Markov chains. Although it achieved better detection results, it had several disadvantages, such as too many states, a high computational cost, poor fault tolerance, and poor generalization. To address these problems, Xiao et al. [4] improved upon Tian et al.’s work [27] by significantly reducing the number of states, which decreased memory consumption and computation. Their model used weighted frequencies of variable-length sequences as a feature. Compared with previous Markov chain-based models, it achieved the best results for SEA Full with a TPR of 90% and FPR of 5%. However, the authors did not report experimental results for PU Truncated Full and Greenberg Truncated Full.

In [29], Geng et al. proposed a computationally cheap model for online masquerade detection. The model exploited N-grams to mine normal user sequences and performed the classification based on square root term frequency-inverse document frequency, which was obtained from natural language processing (NLP). The method obtained a TPR of 93.33% and FPR of 8.8% for SEA Full.

Elmasry et al. [30] conducted massive experiments with various DL models, including a Deep Neural Network (DNN), CNN [13], and Long Short-Term Memory (LSTM). The CNN was leveraged as a text classification model, framing the masquerade detection problem as a text classification task. Although the CNN achieved state-of-the-art results for SEA1v49, PU Enriched Full, Greenberg Enriched Full, PU Truncated Full, and Greenberg Truncated Full on multiple evaluation metrics, the TPR results were not very high for SEA Full (87.7%), PU Truncated Full (81.6%), and Greenberg Truncated Full (83.5%). Thus, achieving a higher TPR for SEA Full, PU Truncated Full, and Greenberg Truncated Full, is still a great challenge. Furthermore, the CNN requires intensive computation and large memory usage. Table 1 lists the best results of the previous work.

2) IDS BASED ON DL MODELS

Compared to traditional ML models, DL methods have shown impressive results for IDS at detecting DoS, probe, U2R, and R2L attacks [31–35]. Kim et al. [31] applied the LSTM architecture and trained the model with the KDD Cup 1999 dataset. They empirically verified that the DL approach is effective for IDS. Yin et al. [32] used RNNs and developed the model RNN-IDS, which outperformed other ML models, including J48, artificial neural network, random forest (RF), and SVM at both binary and multiclass
classification. Meng et al. [33] presented a method based on kernel PCA and LSTM-RNN by integrating data preprocessing, feature extraction, and attack detection into an end-to-end detection system. For the NSL-KDD dataset, their system substantially outperformed several attack detection strategies that use the SVM, neural network, and Bayesian methods. Le et al. [34] explored a model with several components, including an RNN, LSTM, and gated recurrent unit, and discussed which component gave the best result. Abdulhammed et al. [35] employed variational autoencoders and PCA for feature dimensionality reduction. Subsequently, using these features they performed the classification by investigating a variety of ML models, such as RF, Bayesian network, linear discriminant analysis, and quadratic discriminant analysis.

As mentioned in Section I-A, our proposed model extends the Markov chain-based variable-length model of Xiao et al. [4]. Our work significantly differs from the previous work. The main differences are presented as follows. Xiao et al.’s [4] model uses only one feature that is weighted occurrence frequency (of first order), whereas our proposed model introduces two or more features, which are weighted occurrence frequencies of different orders. That leads to an increased number of states (see Sections II-B.5–II-B.7). For instance, for a given $N + 1$ states of the Markov chain in Xiao et al.’s [4] model, our proposed model with two features requires $N^2 + 1$ states to construct the Markov chain. Therefore, we use as the baselines not only Xiao et al.’s [4] model with $N + 1$ states, but also Xiao et al.’s [4] model with $N^2 + 1$ states. In Section IV, we verify empirically that integrating these features boosts our proposed model’s detection performance compared to the baselines. Moreover, as measured by the TPR metric, our proposed model exceeds state-of-the-art CNN [13] and SA-HMM [14] systems. Because our model uses less memory and is computationally less intensive than the state-of-the-art models, it is better suited to real-time masquerade detection.

This article is organized as follows. Section II describes the proposed method. Section III provides the experimental settings. Section IV presents the experimental results and discussion. Section V provides the assessment of the computational and memory complexity. Section VI concludes this article and discusses future work.

II. PROPOSED METHOD

A. MARKOV CHAIN

In this section, we recall some definitions related to the Markov processes from [36].

**Definition 1:** A discrete random process $\{\sigma_n, n \geq 1\}$ is said to be a Markov chain with the state space $\Theta = \{1, 2, \ldots\}$ if
it satisfies the Markov property. That is, for all $n \geq 1$ and
\[
\theta_n \in \Theta \text{ with } 1 \leq m \leq n + 1,
\]
\[
Pr (\sigma_{n+1} = \theta_{n+1} | \sigma_n = \theta_n, \sigma_{n-1} = \theta_{n-1}, \ldots, \sigma_1 = \theta_1)
= Pr (\sigma_{n+1} = \theta_{n+1} | \sigma_n = \theta_n)
\]  
(1)

Definition 2: A Markov chain \( \{\sigma_n, n \geq 1\} \) is said to be homogeneous if the conditional probabilities \( p_{ij}(n) = Pr (\sigma_{n+1} = j | \sigma_n = i) \) are independent of \( n \) for all \( i, j \in \Theta \).

Definition 3: Let \( \{\sigma_n, n \geq 1\} \) be a homogeneous Markov chain and \( p_{ij} = Pr (\sigma_{n+1} = j | \sigma_n = i) \), and \( a_i = Pr (\sigma_1 = i) \) for all \( i, j \in \Theta \).

\[
P = [p_{ij}] \text{ is the matrix of the conditional probabilities } p_{ij} \text{ and is called the (one step) transition (probability) matrix.}
\]

A \( = [a_i] \) is a row vector representing the probability mass function of \( \sigma_1 \) and is called the initial (probability) distribution of the Markov chain.

Theorem 1: Let \( \{\sigma_n, n \geq 1\} \) be a homogeneous stationary Markov chain and \( 1 \leq m \leq n \). Then,

\[
Pr (\sigma_m = \theta_m, \ldots, \sigma_{n-1} = \theta_{n-1}, \sigma_n = \theta_n)
= a_{\theta_m} \cdot p_{\theta_m, \theta_{m+1}} \cdot \ldots \cdot p_{\theta_{n-1}, \theta_n}
\]  
(2)

Fig. 1 presents the architecture of the proposed masquerade detection system. We discuss the details in the next sections.

**B. TRAINING STAGE**

1) PREPROCESSING THE DATA

The training data of a normal user is preprocessed into the sequence of command names \( B = (b_1, \ldots, b_l) \), where \( b_j \in \text{Vocab} \), \( 1 \leq j \leq r \), and \( \text{Vocab} \) is the vocabulary of command names.

2) EXTRACTING VARIABLE-LENGTH SEQUENCES

We extract variable-length sequences and streams of sequences. The definitions are provided below.

Let \( l = \{1(1), 1(2), \ldots, l(W)\} \) be the set of \( W \) lengths of the variable-length sequences, where \( W \in \mathbb{N} (\mathbb{N} = \{1, 2, \ldots\} \) is the set of natural numbers) \( 1(k) \in \mathbb{N} (1 \leq k \leq W) \) and \( l(1) < l(2) < \ldots < l(W) \). We define the variable-length sequences generated from the training data \( B \) as:

\[
S_q^k = (b_q, b_{q+1}, \ldots, b_{q+(k-1)})
\]

where \( 1 \leq q \leq r-l(k)+1 \). The length of the sequence \( S_q^k \) is equal to \( l(k) \). Then, we generate streams of sequences \( \overline{S_1}, \overline{S_2}, \ldots, \overline{S_W} \), where each stream \( \overline{S_k} (1 \leq k \leq W) \) is defined as:

\[
\overline{S_k} = (\overline{S_1^k}, \overline{S_2^k}, \ldots, \overline{S_{r-l(k)+1}^k})
\]

We assumed that \( r > 1(W) - 1 \) to enable the extraction of \( \overline{S_1}, \overline{S_2}, \ldots, \overline{S_W} \). A larger \( W \) means better detection performance, but more memory and computation are required.

3) CALCULATING WEIGHTED OCCURRENCE FREQUENCIES OF DIFFERENT ORDERS

We calculate weighted occurrence frequencies of different orders, which are defined below.

Definition 4: Let \( n_q^k \) be the number of occurrences of a sequence \( \overline{S_q^k} \) in \( \overline{S^k} \) \( (1 \leq k \leq W, 1 \leq q \leq r-l(k)+1) \). Then, the occurrence frequency of \( S_q^k \) in \( S^k \) is defined as follows:

\[
f_q^k = f (\overline{S_q^k}) = n_q^k / (r-l(k)+1)
\]

Definition 5: For a given set of weights \( e_1 = \{e_1(k) \} (1 \leq k \leq W) \), \( 1 \leq e_1 \leq e_2 \leq \ldots \leq e_W \) (W), we calculate the weighted occurrence frequency of the first order as follows:

\[
(\text{wf}_q^k)_1 = e_1(k) \cdot f (\overline{S_q^k})
\]

Note that \( e_1 = e_1(k) \) weights are chosen empirically (see Section III-D) and must satisfy the condition \( 1 \leq e_1(1) \leq e_1(2) \leq \ldots \leq e_1(W) \) condition, which is justified in Remark 1 (Section II-B.6). For brevity, we use the term “weighted frequency” instead of “weighted occurrence frequency” in this article. We generalize Definition 5 to extract more features by introducing the concept of the \( v \)th order weighted frequency with the following definition.

![FIGURE 1. The architecture of the proposed masquerade detection system.](image-url)
**Definition 6:** For a given set of weights $e_v = \{e_v(k)\}$ $(1 \leq k \leq W, 1 \leq v \leq V)$, where $e_v(k)$ is the number of orders, and $V$ is the number of values, we define the $v^{th}$-order weighted frequency using the following recurrent form:

$$
\left(\frac{w f^k_q}{v}\right)_0 = S_q^k
$$

$$
\left(\frac{w f^k_q}{v}\right)_v = e_v(k) \cdot f\left(\left(\frac{w f^k_q}{v-1}\right)\right)
$$

(6)

Here, $f\left(\left(\frac{w f^k_q}{v}\right)\right)$ is computed as follows:

$$
f\left(\left(\frac{w f^k_q}{v}\right)\right) = \frac{n^k_q}{v} / (r - l(k) + 1)
$$

(7)

where $n^k_q$ is the number of occurrences of $\left(\frac{w f^k_q}{v}\right)$ $(1 \leq q \leq r - 1(k) + 1)$ in the stream $\left(\frac{S^k}{v}\right)$ $(1 \leq k \leq W)$ that is defined as:

$$
\left(\frac{S^k}{v}\right) = \left\{\left(\frac{w f^k_q}{v}\right)_1, \left(\frac{w f^k_q}{v}\right)_2, \ldots, \left(\frac{w f^k_q}{v}\right)_{r-l(k)+1}\right\}
$$

(8)

From Eqs. 6, 5, and 4, it is inferred that $n^k_q = (n^k_q)_0$, $S^k_q = (w f^k_q)_0$, $w f^k_q = (w f^k_q)_1$, and $S^k_q \equiv \left(\frac{S^k}{v}\right)$.

For example, if $B = (b_1, \ldots, b_t) = (ci, ci, ci, ci, cd, ls, cd, ls, cd)$, $r = 9$, $W = 2$, $l(1) = 2$, $l(2) = 3$, $e_1(1) = 2, e_1(2) = 3$, $e_2(1) = 2, e_2(2) = 3$, then Table 2 presents the numerical estimate of $(n_q^k)_{v-1}, f\left(\left(\frac{w f^k_q}{v-1}\right)\right), \left(\frac{w f^k_q}{v}\right)_v$ ($v = 1, 2$) for $k = 1$ and $k = 2$, correspondingly, for the same $B.1(l1), l2, e1(1), e1(2)$ with $e2(1) = 2, e2(2) = 3$.

In Section IV, we will empirically verify that incorporating the features $v^{th}$-order weighted frequencies of variable-length sequences boosts the detection performance.

**4) BUILDING THE LIBRARY OF GENERAL SEQUENCES**

First, all different short sequences are extracted from $S_q^k$ and denoted by $\left(\frac{S^k}{1}, \frac{S^k}{2}, \ldots, \frac{S^k}{2}\right)$, where $t(k)$ is the number of different short sequences in $S_q^k$, $(1 \leq k \leq W)$.

Then, for all $k = 1, 2, \ldots, W$, the different short sequences are collected into a set that is called the library of general sequences (LGS) [4] and is denoted by $G = \left(\frac{g^1}{1}, \frac{g^2}{1}, \ldots, \frac{g^1}{t(1)}, \frac{g^2}{1}, \ldots, \frac{g^1}{t(2)}, \ldots, \frac{g^1}{t(W)}, \frac{g^2}{W}, \ldots, \frac{g^i}{t(W)}\right)$. Note that $G$ consists of distinct sequences only.

For example, if $B = (b_1, \ldots, b_t) = (ci, ci, ci, ci, cd, ls, cd, ls, cd, r = 9$, $W = 2$, $l(1) = 2$, $l(2) = 3$, $e_1(1) = 2, e_1(2) = 3$, $e_2(1) = 2, e_2(2) = 3$, then the values $\left(\frac{w f^k_q}{v}\right)_{v=1}$ presented in Table 3 ($k = 1, 2$), we calculate $G^{v=1} = ((ci, ci, (ci, cd, (ls, cd, (ci, ci, (ci, cd, (ls, cd, (ci, ci, (cd, ls, (ci, ci, (cd, ls, (ci, ci, (ci, ci, (ci, cd, (ci, ci, (cd, ls, (ci, ci, (cd, ls, (ci, ci, (ci, ci, (ci, ci, (cd, ls, (ci, ci, (cd, ls)$)

**5) SORTING THE VARIABLE-LENGTH SEQUENCES AND DIVIDING THEM INTO SEVERAL SETS**

For each $v = 1, 2, \ldots, V$ the following two procedures are performed. First, the sequences are sorted in the LGS $G$ such

| $q$ | $b_q$ | $(w f^k_q)_0$ | $(n^k_q)_0$ | $(n^k_q)_1$ | $f\left(\left(\frac{w f^k_q}{v}\right)\right)_1$ | $f\left(\left(\frac{w f^k_q}{v}\right)\right)_2$ |
|-----|-------|----------------|-------------|-------------|--------------------------------|--------------------------------|
| 1   | ci    | (ci, ci)      | 3           | 3/8         | 6/8 (ci, ci, ci)              | 2/7                           |
| 2   | ci    | (ci, ci)      | 3           | 3/8         | 6/8 (ci, ci, ci)              | 2/7                           |
| 3   | ci    | (ci, ci, ci)  | 3           | 3/8         | 6/8 (ci, ci, ci, cd)          | 1/7                           |
| 4   | ci    | (ci, cd)      | 3           | 1/8         | 2/8 (ci, cd, ls)              | 1/7                           |
| 5   | cd    | (cd, ls)      | 2           | 2/8         | 4/8 (cd, ls, cd)              | 2/7                           |
| 6   | ls    | (ls, cd)      | 2           | 2/8         | 4/8 (ls, cd, ls)              | 1/7                           |
| 7   | cd    | (cd, ls)      | 2           | 2/8         | 4/8 (cd, ls, cd)              | 2/7                           |
| 8   | ls    | (ls, cd)      | 2           | 2/8         | 4/8 (ls, cd, ls)              | 1/7                           |
| 9   | cd    | N/A           | N/A         | N/A         | N/A                           | N/A                           |

**TABLE 2.** Numerical estimate that illustrate definitions 4 and 5 ($W = 2, l(1) = 2, l(2) = 3, e_1(1) = 2, e_1(2) = 3$).

**TABLE 3.** Numerical estimate that illustrate definition 6 ($k = 1, W = 2, l(1) = 2, l(2) = 3, e_1(2) = 3, e_2(1) = 2, e_2(2) = 3$).
that their \(v^\text{th}\) -order weighted frequencies are in descending order, and \(G\) is denoted by \(G^v\). Second, we divide \(G^v\) into \(N\) sets to build \(\Lambda^v = \{\Lambda_0^v, \ldots, \Lambda_{N-1}^v\}\) as follows. The number of elements in \(G^v\) is denoted by \(|G^v| = |G| = t(1) + \ldots + t(W)\), and \(\alpha = |H/N\) denotes the greatest integer not larger than \(H/N\). Let \(\beta = \alpha + 1, h = H - Na\). Note that \(H = h\beta + (N - h)\alpha\). Thus, we split \(G^v\) into \(N\) sets such that the first \(h\) sets consist of \(\beta\) elements, and the last \((N - h)\) sets contain \(\alpha\) elements. We define \(\Lambda_0^v = \{g^v_1, \ldots, g^v_{\beta}\}\), \(\Lambda_h^v = \{g^v_{(h-1)\beta+1}, \ldots, g^v_{h\beta}\}\), and \(\Lambda^v_{h-1} = \{g^v_{h\beta+1}, \ldots, g^v_{(h-1)\beta+\alpha}\}\) to produce \(\Lambda^v = \{\Lambda_0^v, \ldots, \Lambda_{N-1}^v\}\). This splitting process was introduced by Xiao et al. [4], but whereas their splitting mechanism works for \(V = 1\) only, whereas ours generates \(\Lambda^v\) sets for each \(v = 1, 2, \ldots, V\).

Note that \(\Lambda^v\) and \(G\) contain only the same short sequences.

### 6) DEFINING STATES BY MINING BEHAVIORAL PATTERNS

We first call Algorithm 1 (Defining states), which in turn calls Algorithm 2 (Mining behavioral patterns). Training sequence \(B = (b_1, \ldots, b_p)\) is passed as \(s = (s_1, s_2, \ldots, s_r)\) to Algorithm 2, which returns states \(\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_M)\). Below, we present Algorithm 1 and Algorithm 2.

#### Remark 1: Algorithm 1 is similar to Algorithm 1 of Xiao et al. [4] (Defining behavioral patterns). The difference is that our algorithm has an additional loop \((\text{for } v = 1, \ldots, V \text{ iterations do})\) to mine behavioral patterns based on each feature, i.e., the \(v^\text{th}\) -order weighted frequency (see line 4, Algorithm 1). Note that \(\text{getWeightedFrequency}_v\) (shortSequence) returns the \(v^\text{th}\) -order weighted frequency of shortSequence. Note that for \(\forall k_1, k_2 (1 \leq k_1 < k_2 \leq W)\), \(n^k_{1} \geq n^k_{2}\) \((1 \leq j = r - 1(W) + 1)\), where \(n^k_{1}\) is the number of occurrences of the \(s_1 = (s_j, \ldots, s_{j + \lceil k_1 \rceil - 1})\) in \(S^{k_1}\), and \(n^k_{2}\) is the number of occurrences of the \(s_2 = (s_j, \ldots, s_{j + \lceil k_2 \rceil - 1})\) in \(S^{k_2}\). Hence, the \(e_1\) weights need to satisfy the condition \(1 \leq e_1 (1) \leq e_1 (2) \leq \ldots \leq e_1 (W)\) not to let shortSequenceMax at position \(j\) be shortSequence with the smallest length. If shortSequence \(\notin G\), then \(\text{getWeightedFrequency}_v\) (shortSequence) returns 0. Thus, for each short sequence of the training data, the method returns a value greater than 0. However, this is not always the case for the detection data. When the value is 0, Algorithm 1 returns an empty \(g\).

#### Remark 2: The method \(\Lambda^v, \text{getIndex}(g_v)\) returns the index of the set in \(\Lambda^v\) to which \(g_v\) belongs. For instance, if \(g_v \in \Lambda^v_3\), then \(\Lambda^v, \text{getIndex}(g_v)\) returns 3. If \(g\) is not empty, then \(\sigma_m\) is computed as the sum of \(\sigma_m\) multiplied by the coefficients \(N^v-v; \sigma_m = \sigma_{m1} \cdot N^{v-1} + \sigma_{m2} \cdot N^{v-2} + \ldots + \sigma_{mN}\). Otherwise, we assign the state \(N\) to \(\sigma_m\) (line 20). This algorithm generalizes Algorithm 2 of Xiao et al. [4] (Defining states) by introducing an additional loop with \(V\) iterations to compute the state \(\sigma_m\) (see line 12, Algorithm 2). Note that \(\sigma_m \in \{0, 1, \ldots, N^v-1, N^v\}\) and therefore, the number of different states is \(N^v + 1\). By contrast, in Xiao et al.'s [4] model, the number of distinct states is \(N + 1\).
Algorithm 2 Defining states
1. Input: $\Lambda = (\Lambda^1, \Lambda^2, \ldots, \Lambda^V)$, where each $\Lambda^v = (\Lambda_1^v, \Lambda_2^v, \ldots, \Lambda_{N-1}^v)$ ($1 \leq v \leq V$);
2. commands $s = (s_1, s_2, \ldots, s_r)$; set of lengths $l = \{l(1), l(2), \ldots, l(W)\}$;
3. Output: states $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_M)$
4. Initialization: states $\sigma = ()$, $m = 1$, $j = 1$, $NV = NV^v$
5. while ($j \leq r - 1 (W + 1)$) do
6. $g = \text{mineBehavioralPattern}(l, s, j)$ (using Algorithm 1)
7. iMax = 1
8. if (!g.empty()): // if g is not empty
9. $\sigma_m = 0$
10. for $v = 1, \ldots, V$ iterations do
11. $\sigma_{mv} = \Lambda^v, \text{getIndex}(g_v)$
12. $\sigma_{m\sigma} = \sigma_m + \sigma_{mv} NV^{-v}$
13. if (g_v.length() > iMax): iMax = g_v.length()
14. end if
15. end for
16. end if
17. else
18. if (g𝐗 == empty()) // if g is not empty
19. $\sigma_{m\sigma} = NV$
20. end if
21. if (iMax == g_v.length())
22. $\sigma_m, m = m + 1$
23. end if
24. j = j + 1
25. end while
26. return $\sigma$

Algorithm 3 Building probability distributions
1. Input: states $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_M)$; number of sets $N$
2. Output: initial probability distribution $A$, transition probability distribution matrix $P$
3. Initialization: $NV = NV^v$, initialize the states’ occurrence time vector $Y \in R^{NV + 1}$
4. probability distribution vector $A \in R^{NV + 1}$
5. probability stream matrix $P \in R^{(NV + 1) \times (NV + 1)}$ with zeros.
6. for $m = 1, \ldots, M$ iterations do
7. $Y_{\sigma_m} = Y_{\sigma_m} + 1$ // increment states’ occurrence times
8. if ($m < M - 1$):
9. // increment the number of transitions from state $\sigma_m$ to state $\sigma_{m+1}$
10. $P_{\sigma_m, \sigma_{m+1}} = P_{\sigma_m, \sigma_{m+1}} + 1$
11. end if
12. end for
13. end for
14. for $m = 0, \ldots, NV$ iterations do
15. $A_{\sigma_0} = (0, M / M) // normalize the occurrence times of each state
16. for $n = 0, \ldots, NV$ iterations do
17. if ($Y_m \neq 0$):
18. // normalize the number of transitions from state $\sigma_m$ to $\sigma_{m+1}$
19. $P_{\sigma_m, \sigma_{m+1}} = P_{\sigma_m, \sigma_{m+1}} / Y_m$
20. end if
21. end for
22. end for
23. $P_{NV, NV} = 1$
24. return $A, P$

7) BUILDING PROBABILITY DISTRIBUTIONS
We build probability distributions for the defined states $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_M)$ using Algorithm 3.

Remark 3: Note that none of the states built on training data is NV = NV^v therefore $A_{NV} = 0, P_{NV, \sigma_m} = 0$, $P_{\sigma_m, NV} = 0, \sigma_m, NV = 0$, $m = 1, \ldots, M$. If $\sigma_m = NV$, then $Pr(\sigma_m = \theta_m, \ldots, \sigma_n = \theta_n) = 0$. In Algorithm 3, we use $NV + 1$ distinct states instead of the $N + 1$ states used by Xiao et al. [4]. Consequently, in our proposed model the initial probability distribution vector $A$ and state transition probability matrix $P$ have $NV + 1$ and $(NV + 1)^2$ dimensions, correspondingly, in place of $N + 1$ and $(N + 1)^2$ in Xiao et al.’s [4] model, respectively. This is the only difference between our proposed Algorithm 3 and Algorithm 3 of Xiao et al. [4] (Building probability distributions).

C. DETECTION STAGE
The detection stage consists of the following steps borrowed from Xiao et al. [4].

1) PREPROCESSING THE DATA
The testing data of an audited user is preprocessed into the sequence of command names $B = (b_1, b_2, \ldots, b_{\tilde{r}})$, where $\tilde{r}$ is the number of command names. We assume that $\tilde{r} > 1 (W + w - 1$ in order to generate variable-length sequences with the maximum length $1 (W)$ over the window $w$.

2) DEFINING STATES BY MINING BEHAVIORAL PATTERNS
Algorithm 2 (Section II-B.6) is applied to the detection data $B = (b_1, b_2, \ldots, b_{\tilde{r}})$ to yield the states $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_M)$, assuming that $\tilde{r} > 1 (W) - 1$.

3) CALCULATING THE PROBABILITIES OF SHORT STATE SEQUENCES
For $m = 1, \ldots, \tilde{m} - u + 1$ short state sequences $\sigma_m = [\sigma_m, \sigma_{m+1}, \ldots, \sigma_{m+u-1}]$ are built, where $u$ is the length of a short state sequence. The probabilities of the sequences are calculated according to Theorem 1:

$$Pr(\sigma_m) = a_{\sigma_m} P_{\sigma_m, \sigma_{m+1}} P_{\sigma_{m+1}, \sigma_{m+2}} \cdots P_{\sigma_{m+u-2}, \sigma_{m+u-1}}$$

In particular, when $u = 2$, we set $i = \sigma_m$ and $j = \sigma_{m+1}$; then, $Pr(\sigma_m) = a_{\sigma_m} P_{\sigma_{m+1}}$. Once the probability of each short sequence is calculated, then all of them can
be used to produce the probability stream denoted by
\( \Pr(\sigma_1), \Pr(\sigma_2), \ldots, \Pr(\sigma_{\tilde{M}-u+1}) \).

4) COMPUTING THE DECISION VALUES

We assumed that, in the detection stage, the commands of a misbehaving user may match the commands of a normal user for a short period of time but will eventually differ over a longer period [4], [6], [8]. Therefore, we can make decisions about the type of behavior by considering \( w \) short state sequences, where \( w \) is the window size.

Furthermore, some low probabilities may be due to a coincidence. To stop these probabilities from producing large values when summed over the window \( w \), assuming that \( \tilde{r} > 1(\tilde{W}) + w - 1 \), the decision value \( D(n) \) is calculated in terms of the probabilities of the short state sequences as follows [4]:

\[
D(n) = \frac{1}{w} \sum_{m=n-w+1}^{n} (\Pr(\sigma_m) - \eta) \tag{10}
\]

\( w \leq n \leq \tilde{M}-u+1 \) and \( \eta \) is a predefined probability threshold.

5) CLASSIFYING THE DECISION VALUES

Suppose that the decision threshold is \( \tau \). An audited user’s \( n \)th behavior is classified as normal or masquerade based on \( D(n) \). If \( D(n) \geq \tau \), then the \( n \)th behavior is ranked as normal; otherwise, it is ranked as masquerade. Note that the \( n \)th behavior consists of \( \sigma_{n-w+1}, \sigma_{n-w+2}, \ldots, \sigma_n \) short state sequences. Normally, a higher \( \tau \) classification threshold catches more masquerade behavior but may increase the FPR. In contrast, a lower threshold \( \tau \) may decrease not only the FPR but also the TPR. Thus, the choice of the threshold \( \tau \) value depends on the application.

D. OUTLINE OF THE PROPOSED MODEL

The training stage of the proposed method consists of the following steps:

- Preprocessing the data
- Extracting variable-length sequences
- Calculating weighted occurrence frequencies of different orders
- Building the LGS
- Sorting the variable-length sequences and dividing them into several sets
- Defining states by mining behavioral patterns
- Building probability distributions

The detection stage consists of the following steps:

- Preprocessing the data
- Defining states by mining behavioral patterns
- Calculating the probabilities of short state sequences
- Computing the decision values
- Classifying the decision values

III. EXPERIMENTS

A. DATASET

We conducted experiments on three publicly available and free UNIX command-line-based datasets: Schonlau (http://www.schonlau.net/intrusion.html), PU(http://kdd.ics.uci.edu/databases/UNIX_user_data/UNIX_user_data.html), and Greenberg (http://saul.cpsc.ucalgary.ca/pmwiki.php) with the permission of its owner. Each dataset consists of text files, each containing commands from one user. Thus, there were no real masqueraders in the dataset. The experiments were performed on each dataset with the truncated data configuration only. For each data setting, we generated a training set \( B = (b_1, \ldots, b_r) \) for the training stage (see Section II-B, Section II-B.1), and a testing set \( \tilde{B} = (\tilde{b}_1, \tilde{b}_2, \ldots, \tilde{b}_\tilde{r}) \) for the detection stage (see Section II-C, Section II-C.1), as described below.

1) SCHONLAU DATASET

Schonlau et al. [9] introduced a free UNIX command-line-based dataset called SEA with the truncated data configuration only. The UNIX acct audit tool was used to collect commands from NU SEA = 50 different users for several months. We used the following data settings proposed by Schonlau et al. [9], which we named “SEA Full” for NU SEA = 50, and “SEA Shrunk” for NU SEA = 4. For each user, the first 5000 commands yielded the training set, and the next 10,000 commands built the test set. Each user’s test set is contaminated with the commands of other users at random positions. The positions were given at (http://www.schonlau.net/masquerade/masquerade_summary.txt) in .txt format, which we used to generate the normal and masquerade test sets.

2) PU DATASET

The PU dataset was introduced by Lane and Brodley [10]; the data were taken from the tcsh history files and stored with the enriched data configuration. The dataset contains session information for each of NU PU = 8 users. We converted the dataset into the truncated data configuration and used the following data settings proposed by Lane and Brodley [10], which we named “PU Truncated Full” for NU PU = 50, and “PU Truncated Shrunk” for NU PU = 4. For each user, the training set was the first 1500 commands, and the test set was the next 500 commands. We also took 500 commands of the other (NU PU − 1) users starting from the 1501th command, resulting in 500 × (NU PU − 1) commands for the masquerade test set.

3) GREENBERG DATASET

Greenberg’s dataset [11] was collected through the csh tool and is stored with the enriched data configuration. There were 168 UNIX users in the dataset divided into four groups: novice programmers, experienced programmers, computer scientists, and non-programmers. We converted the dataset into the truncated data configuration and used the following data settings proposed by Maxion [12]. We selected users with 2000 − 5000 commands. NU Greenberg = 50 users satisfied the condition. Then, for each user of NU Greenberg users, the first 2000 commands were used for the experiments.
TABLE 5. Datasets.

| Dataset                | Year | Tools for extraction | # users (NU) | # train set per user | # test set per user |
|------------------------|------|----------------------|-------------|----------------------|---------------------|
|                        |      |                      |             | Normal | Masquerade | Total | Normal | Masquerade | Total |
| SEA Shrunken           | 2001 | act                  | 4           | 5000   | 0          | 5000  | 9538   | 462       | 10000 |
| SEA Full               | 2001 | act                  | 50          | 5000   | 0          | 5000  | 9538   | 462       | 10000 |
| PU Truncated Shrunken  | 1997 | tcsh                 | 4           | 1500   | 0          | 1500  | 500    | 1500      | 2000  |
| PU Truncated Full      | 1997 | tcsh                 | 8           | 1500   | 0          | 1500  | 500    | 3500      | 4000  |
| Greenberg Truncated Shrunken | 1988 | csh                  | 4           | 1000   | 0          | 1000  | 1000   | 300       | 1300  |
| Greenberg Truncated Full| 1988 | csh                  | 50          | 1000   | 0          | 1000  | 1000   | 300       | 1300  |

The first 1000 commands were used to create the training set; the next 1000 commands yielded the normal test set. We chose 25 users from the remaining of the 118 users to act as masqueraders. From these users, we randomly selected 300 commands to be the masquerade test set for each of the NUGreenberg users. Note that each set of 300 commands was distinct. Thus, each user’s training set consisted of 1000 commands. The normal test set contained 1000 commands, and the masquerade test set contained 300 commands. We named these data settings “Greenberg Truncated Full” for NUGreenberg = 50, and “Greenberg Truncated Shrunken” for NUGreenberg = 4. Table 5 briefly describes the datasets with the associated data settings.

B. BASELINES

Xiao et al.’s [4] Markov chain-based variable-length method (V = 1) was denoted as “Markov.” Our proposed Markov chain-based variable-length model with both first- and second-order weighted frequencies (V = 2) was denoted as “MarkovF2.” Note that, for a given number of N sets, Markov uses (N + 1) states. In contrast, our proposed MarkovF2 uses N^2 + 1 states (see Section II-B.6). Therefore, we also used Xiao et al.’s [4] Markov model with N^2 + 1 states (i.e., N^2 sets) as an additional baseline and denoted it as “MarkovN2.” Thus, the proposed model is MarkovF2; both baselines Markov and MarkovN2 are from Xiao et al. [4].

C. EVALUATION METRICS

To assess the performance of our model, we used different evaluation metrics such as the TPR, FPR, ROC, threshold variance, computational complexity, and memory complexity. Let D be a given dataset with specified data settings, and NU be the number of users. Then for each user i’s masquerade test set (useri ∈ D, 1 ≤ i ≤ NU), the numbers of true positive and false negative predictions are denoted by TP_i and FN_i, respectively, whereas, for user i’s normal test set, the numbers of false positive and true negative predictions are denoted by FP_i and TN_i, respectively. User i’s true positive rate TPR_i is

\[ TPR_i = \frac{TP_i}{TP_i + FN_i} \]  \hspace{1cm} (11)

A higher TPR_i indicates better masquerade detection. By contrast, a lower FPR_i indicates a better classifier for normal test data, where

\[ FPR_i = \frac{FP_i}{FP_i + TN_i} \]  \hspace{1cm} (12)

For D, we calculate TPR_{mean} = ∑_{i=1}^{NU} TPR_i/NU and FPR_{mean} = ∑_{i=1}^{NU} FPR_i/NU. We define ROC as the plot of TPR_{mean} against FPR_{mean} for each threshold value τ from −1 to 1 in increments of 0.002.

For each user_i (user_i ∈ D, 1 ≤ i ≤ NU), we denote a threshold value τ by threshold_{optimal} that yields the maximum TPR (TPR_{max}) value with the constraint that the corresponding FPR ≤ 0.1. We denote this FPR_i as FPR_{optimal}. Then, the TPR is defined as follows.

\[ TPR = \sum_{i=1}^{NU} TPR_i^{\text{max}} / NU \]  \hspace{1cm} (13)

and the FPR is defined as follows.

\[ FPR = \sum_{i=1}^{NU} FPR_i^{\text{optimal}} / NU \]  \hspace{1cm} (14)

Threshold variance is calculated through the following equation.

\[ \text{threshold variance} = \frac{\sum_{i=1}^{NU} (\text{threshold}_i - \text{threshold mean})^2}{NU} \]  \hspace{1cm} (15)

where threshold mean is the average threshold value, that is

\[ \text{threshold mean} = \frac{\sum_{i=1}^{NU} \text{threshold}_i}{NU} \]  \hspace{1cm} (16)

Lower threshold variance indicates better generalization. We consider threshold mean to be a recommended threshold value for D.

D. EXPERIMENTAL SETUP

The experiments were implemented in the Java programming language. For reproducibility, the code was deposited on GitHub (https://github.com/kazaros91/Masquerade-Detection.
The experiments were conducted on a MacBook Pro with an Intel Core i7 processor, 2.9 GHz processor, four cores, and 16 GB RAM. We performed a greedy search with the hyperparameters presented in Table 6. Because of space limitations, we only provided the best results obtained through the greedy search.

### IV. RESULTS AND DISCUSSION

This section is organized as follows. Section IV-A presents the comparison of our model to the baselines using Shrunk data settings. Section IV-B presents the comparison of our model to the baselines using Full data settings. Section IV-C presents the effect of the window size and number of sets on the detection performance using Full data settings. Section IV-D presents the comparison of our model to state-of-the-art models, such as CNN [13] and SA-HMM [14], using Full data settings.

#### A. RESULTS WITH SHRUNK DATA SETTINGS

For SEA Shrunk, the best hyperparameters were $W = 3, l = \{1, 2, 3\}, e_1 = \{1, 2, 3\}, e_2 = \{1, 2, 3\}, N = 2, w = 81, \eta = 0.1$. The results are presented in Table 7, and the ROC curves are presented in Fig. 2.

Table 7 demonstrates that, for SEA Shrunk, the proposed MarkovF2 obtains noticeably better TPR, FPR and threshold variance than Markov and MarkovN2.

For Greenberg Truncated Shrunk, the best hyperparameters were $W = 3, l = \{1, 2, 3\}, e_1 = \{3, 4, 5\}, e_2 = \{1, 3, 5\}, N = 2, w = 81, \eta = 0.1$. Table 9 presents the results, and Fig. 3 shows the ROC curves.

Table 9 demonstrates that, for Greenberg Truncated Shrunk, the proposed MarkovF2 obtains substantially better TPR, FPR and threshold variance than Markov and MarkovN2.

For PU Truncated Shrunk, the best hyperparameters were $W = 3, l = \{1, 2, 3\}, e_1 = \{1, 2, 3, 4\}, e_2 = \{1, 3, 5\}, N = 2, w = 81, \eta = 0.1$. Table 8 presents the results, and Fig. 4 shows the ROC curves.

Table 8 demonstrates that, for PU Truncated Shrunk, the proposed MarkovF2 achieves perfect TPR = 1 and FPR = 0, while also having significantly better threshold variance than Markov and MarkovN2.

For Greenberg Truncated Shrunk, the proposed MarkovF2 outperforms Markov and MarkovN2 at ROC curves. Markov is better than MarkovN2 in terms of ROC curves.
For Greenberg Truncated Shrunken, Fig. 4 shows that the 
FPR_{mean} is in [0, 0.06], the proposed MarkovF2 is superior 
to Markov and MarkovN2 at ROC curves. When FPR_{mean} is 
in range [0,0.06] MarkovF2 still outperforms the baselines 
but not very noticeably.

In sum, for all Shrunken data settings (SEA Shrunken, 
PU Truncated Shrunken, Greenberg Truncated Shrunken), 
Tables 7–9 and Figures 2–4 demonstrate that the proposed 
MarkovF2 model substantially outperforms the baselines 
(Markov and MarkovN2) at several metrics, including TPR, 
FPR, threshold_variance, and ROC curves. Thus, incorpo-
rating a higher order weighted frequency indeed boosts the 
detection performance, as claimed in Section 1-A.

### B. RESULTS WITH FULL DATA SETTINGS

For SEA Full the best hyperparameters were $W = 3, l = 
\{1, 2, 3\}, e_1 = \{1, 2, 3\}, e_2 = \{1, 2, 3\}, N = 2, w = 81, \eta = 0.1$. Table 10 presents the results, and Fig. 5 presents 
the ROC curves.

Table 10 demonstrates that, for SEA Full, compared to 
Markov and MarkovN2, the proposed MarkovF2 obtains 
remarkably better TPR and threshold_variance, as well as 
oticeably lower FPR.

For SEA Full, Fig. 5 demonstrates that MarkovF2 has su-
pe rior TPR_{mean} than the baselines (Markov and MarkovN2) by 
significant margins, when the FPR_{mean} is in [0, 0.06]. When 
the FPR_{mean} is in [0.06, 0.17], MarkovF2 still outperforms 
the baselines, but not very noticeably.

For PU Truncated Full, the best hyperparameters were $W = 3, l = 
\{1, 2, 3\}, e_1 = \{1, 2, 3\}, e_2 = \{1, 3, 5\}, N = 2, w = 81, \eta = 0.1$. Table 11 presents the results, and Fig. 6 
presents the ROC curves.

Table 11 demonstrates that, for PU Truncated Full, the pro-
posed MarkovF2 is superior to Markov and MarkovN2 in 
terms of TPR, FPR, and threshold_variance by significant 
margins.

Fig. 6 shows that, for PU Truncated Full, Markov and 
MarkovN2 have higher TPR_{mean} values than MarkovF2 when 
the FPR_{mean} is in [0.04, 0.06]. For other FPR_{mean} values, 
the proposed MarkovF2 substantially outperforms the base-
lines.

For Greenberg Truncated Full, the best hyperparam-
eters were $W = 3, l = \{1, 2, 3\}, e_1 = \{3, 4, 5\}, e_2 = \{1, 3, 5\}, N = 2, w = 81, \eta = 0.1$. Table 12 presents the 
results, and Fig. 7 presents the ROC curves.

Table 12 demonstrates that, for Greenberg Truncated Full, 
the proposed MarkovF2 significantly outperforms Markov

### TABLE 9. Results for Greenberg Truncated Shrunk.

| Model      | TPR   | FPR   | Threshold variance | Threshold_mean |
|------------|-------|-------|--------------------|---------------|
| Markov     | 0.856 | 0.024 | 0.245              | 0.332         |
| MarkovN2   | 0.815 | 0.048 | 0.227              | 0.278         |
| MarkovF2   | 0.940 | 0.018 | 0.096              | 0.046         |

### TABLE 10. Results for SEA Full.

| Model      | TPR   | FPR   | Threshold_variance | Threshold_mean |
|------------|-------|-------|--------------------|---------------|
| Markov     | 0.901 | 0.049 | 0.048              | 0.170         |
| MarkovN2   | 0.887 | 0.048 | 0.056              | 0.123         |
| MarkovF2   | 0.953 | 0.043 | 0.025              | -0.055        |

### TABLE 11. Results for PU Truncated Full.

| Model      | TPR   | FPR   | Threshold_variance | Threshold_mean |
|------------|-------|-------|--------------------|---------------|
| Markov     | 0.803 | 0.057 | 0.081              | 0.044         |
| MarkovN2   | 0.800 | 0.056 | 0.058              | 0.001         |
| MarkovF2   | 0.843 | 0.049 | 0.041              | -0.022        |

### FIGURE 4. ROC curves for Greenberg Truncated Shrunk.

### FIGURE 5. ROC curves for SEA Full.

### FIGURE 6. ROC curves for PU Truncated Full.
and MarkovN2 in terms of TPR and threshold variance. Furthermore, MarkovF2 has noticeably lower FPR than the baselines.

**TABLE 12. Results for Greenberg Truncated Full.**

| Model   | TPR  | FPR  | Threshold variance | Threshold mean |
|---------|------|------|--------------------|----------------|
| Markov  | 0.875| 0.048| 0.218              | 0.301          |
| MarkovN2| 0.854| 0.053| 0.212              | 0.267          |
| MarkovF2| 0.887| 0.045| 0.085              | 0.033          |

Fig. 7 shows that, for Greenberg Truncated Full, Markov, MarkovN2, and MarkovF2 have similar ROC curves. For some FPR mean values, Markov demonstrates a slightly higher TPR mean, whereas MarkovF2 outperforms Markov and MarkovN2 for other FPR mean values but not by a significant amount.

In sum, for all Full data settings (SEA Full, PU Truncated Full, Greenberg Truncated Full), Tables 10–12 and Figures 5–7 show that the proposed MarkovF2 significantly outperforms the baselines at several metrics, including TPR, FPR, threshold variance, and ROC curves. Thus, incorporating a higher order weighted frequency indeed enhanced the detection performance, as claimed in Section 1-A.

**C. EFFECT OF WINDOW SIZE AND NUMBER OF SETS ON PERFORMANCE**

Recall that w is the window size and N is the number of sets. Figs. 8–10 illustrate the impacts of w and N on the TPRs of all models (Markov (a), MarkovN2 (b), and MarkovF2 (c)) for SEA Full, PU Truncated Full, and Greenberg Truncated Full. We identify the following dependencies: for N = 2, 3, 4, the TPR values of all models increase with window size w. For all window sizes (w = 21, 36, 51, 81), the TPR values of all models decrease with increasing N. Thus, the highest TPR is achieved at w = 81 and N = 2.

**D. COMPARISON TO THE STATE-OF-THE-ART MODELS**

Table 13 presents the best results for the proposed model (MarkovF2), the baselines (Markov and MarkovN2), and the
state-of-the-art models (CNN [13] and SA-HMM [14]) for SEA Full, PU Truncated Full, and Greenberg Truncated Full. Although the CNN has a better FPR than all the other models, MarkovF2 has the best TPR. What is more important is that the MarkovF2 achieves significant improvement in terms of TPR on the datasets with the truncated data configuration, which means that MarkovF2 is more effective when data samples contain limited information.

V. COMPUTATIONAL COMPLEXITY AND MEMORY COMPLEXITY

Table 14 presents the computational and memory complexity of the models. (We did not analyze the computational and memory complexity of SA-HMM because HMM-based methods inherently have a higher computational and memory complexity.) Table 15 provides the parameter values used to evaluate the computational and memory complexity. The numerical estimate of the complexity presented in Table 16 assume that the memory unit is 4 bytes. Table 16 indicates that compared to CNN [13], MarkovF2 requires approximately $10^9$ times fewer ops in the training stage, approximately $10^{10}$ times fewer ops in the detection stage, and approximately $10^6$ times fewer memory units. The details are provided in Sections IV-A and IV-B.

A. MARKOV CHAIN-BASED MODELS

Recall that $H$ is the number of distinct sequences in the LGS, i.e., the LGS size. The computational complexity in the training stage can be assessed as follows. Extracting variable-length sequences needs time $O(W \times r)$, calculating weighted occurrence frequencies requires $O(V \times r)$ ops, and building the LGS takes $O(H)$ ops. Sorting variable-length sequences and dividing them into $V$ states requires $O(V \times H \times \log H)$ ops. Then, defining states by mining behavioral patterns needs $O(V \times L \times \log H)$ ops, where $\log H$ is the search time to find a sequence in the sorted sets ($1 \leq v \leq V$) (see Algorithm 1 and Algorithm 2) and $L$ is the average length of variable-length sequences. Finally, building probability distributions involves $N^V + 1 + (N^V + 1)^2 = O(N^{2V})$ ops (see Algorithm 3). In total, the computational complexity of the Markov chain-based model in the training stage is

$$O(W \times r) + O(V \times r) + O(H) + O(V \times H \times \log H) + O(V \times L \times \log H) + O(N^{2V}) = O(V \times H \times \log H).$$

After an audited user’s data are preprocessed in the detection stage, the computational complexity can be estimated as follows. Defining states by mining behavioral patterns requires $O(V \times L \times \log H)$ ops, calculating the probabilities of short state sequences takes time $O(1)$, computing decision values requires $O(w)$ ops, and classifying the decision values needs $O(1)$ ops. Thus, the computational complexity of the Markov chain-based model in the detection stage is

$$O(V \times L \times \log H) + O(N^V + 1)^2 + O(w) + O(1) = O(V \times L \times \log H).$$
The LGS size is \( H \), the initial probability distribution vector \( A \) vector has size \( N^V + 1 \), and the transition probability distribution matrix \( P \) matrix has size \( (N^V + 1)^2 \). Thus, the memory complexity is given by \( H + N^V + 1 + (N^V + 1)^2 = O(H + N^{2V}) \). The memory complexity of Markov (\( V = 1 \)) is therefore \( O(H + N^2) \), that of MarkovN2 (\( V = 1 \)) is \( O(H + N^4) \), and that of our MarkovF2 (\( V = 2 \)) is \( O(H + N^6) \). The best results were achieved when \( N=2 \), which is very small.

**TABLE 13.** Best results.

| Dataset             | Model     | TPR  | FPR  | Threshold_variance | Threshold_mean |
|---------------------|-----------|------|------|--------------------|----------------|
| SEA Full            | Markov    | 0.901| 0.049| 0.048              | 0.171          |
|                     | MarkovN2  | 0.887| 0.048| 0.056              | 0.123          |
|                     | MarkovF2  | 0.953| 0.043| 0.025              | -0.055         |
| SA-HMM              | 0.941     | 0.042| N/A  | N/A                | N/A            |
| CNN [13]            | 0.877     | 0.005| N/A  | N/A                | N/A            |
| PU Truncated Full   | Markov    | 0.803| 0.057| 0.081              | 0.044          |
|                     | MarkovN2  | 0.800| 0.056| 0.058              | 0.001          |
|                     | MarkovF2  | 0.843| 0.049| 0.041              | -0.022         |
| Greenberg           | 0.816     | 0.015| N/A  | N/A                | N/A            |
| Truncated Full      | Markov    | 0.875| 0.048| 0.218              | 0.301          |
|                     | MarkovN2  | 0.854| 0.053| 0.212              | 0.267          |
|                     | MarkovF2  | 0.887| 0.045| 0.085              | 0.033          |
|                     | CNN [13]  | 0.835| 0.010| N/A                | N/A            |

**TABLE 14.** Computational and memory complexity.

| Model           | Computational complexity | Memory complexity |
|-----------------|--------------------------|-------------------|
|                 | Training stage           | Detection stage   | Training and detection stages |
| Markov (\( V = 1 \)) | \( O(H \times \log H) \) | \( O(L \times \log H) \) | \( O(H + N^2) \) |
| MarkovN2 (\( V = 1 \)) | \( O(H \times \log H) \) | \( O(L \times \log H) \) | \( O(H + N^4) \) |
| MarkovF2 (\( V = 2 \)) | \( O(2 \times H \times \log H) \) | \( O(2 \times L \times \log H) \) | \( O(H + N^4) \) |
| CNN [13] | \( O(540 \times r \times K) \times C^2 + 270 \times z^2 \) | \( O(384 \times K \times C^2 + 3 \times z^2) \) | \( O(192 \times z^2) \) |

**TABLE 15.** Parameter values for assessment of the computational and memory complexity.

| \( O(H) \) | \( N \) | \( O(r) \) | \( O(K) \) | \( O(C) \) | \( O(z) \) |
|------------|---------|-----------|-----------|-----------|-----------|
| 3500       | 2       | 5000      | 7         | 1024      | 2048      |

**TABLE 16.** Numerical estimate of the computational and memory complexity.

| Model           | Computational complexity (ops) | Memory complexity (bytes) |
|-----------------|-------------------------------|--------------------------|
|                 | Training stage | Detection stage | Training and detection stages |
| Markov (\( V = 1 \)) | \( 4.2 \times 10^4 \) | \( 3.6 \times 10 \) | \( 14.01 \times 10^3 \) |
| MarkovN2 (\( V = 1 \)) | \( 4.2 \times 10^4 \) | \( 3.6 \times 10 \) | \( 14.01 \times 10^3 \) |
| MarkovF2 (\( V = 2 \)) | \( 8.4 \times 10^4 \) | \( 7.2 \times 10 \) | \( 14.06 \times 10^3 \) |
| CNN [13] | \( 1.98 \times 10^{13} \) | \( 2.2 \times 10^{12} \) | \( 14.48 \times 10^9 \) |

**B. CNN**

Let \( N_{conv} = 6 \) be the number of convolutional layers of the CNN [13], \( K \) be the kernel size (\( O(K) = 7 \)), and \( \text{InChannels} = \text{OutChannels} = C = 1024 \), where \( \text{InChannels} \) and \( \text{OutChannels} \) are the numbers of input and output channels, respectively, in each convolutional layer. \( N_{fully} = 3 \) is the number of fully connected (FC) layers, and \( z = 2048 \) is the number of neurons in each FC layer. Then, for the CNN [13], the computational complexity in the training stage can be determined as follows: each convolutional layer performs the convolution operation and pooling, which in sum requires \( O(r - K + 1) \times K \times \text{InChannels} \times \text{OutChannels} \) ops. Each FC layer performs \( O(z^2) \) ops. Thus, each forward pass requires \( F = O(N_{conv} \times r \times K \times C^2 + N_{fully} \times z^2) \) ops. Each backward pass needs \( B = 2 \times F \) ops, as argued by He and Sun [37]. In total, the computational complexity in the training stage of the CNN [13] is \( T = O(E \times (F + B)) = O(E \times 3 \times (N_{conv} \times r \times K \times C^2 + N_{fully} \times z^2)) \), where \( E = 30 \) is the number of epochs. Thus, \( T = O(540 \times r \times K \times C^2 + 270 \times z^2) \).

The detection stage consists of the forward pass only, for which the computational complexity is \( F = O(N_{conv} \times r \times K \times C^2 + N_{fully} \times z^3) = O(6 \times r \times K \times C^2 + 3 \times z^3) \).
Each convolutional layer of the CNN [13] is a weight tensor in $\mathbb{R}^{\text{batch size} \times 1 \times K \times \text{InChannels} \times \text{OutChannels}}$, where $\text{batch size} = 64$ and $\text{InChannels} = \text{OutChannels} = C = 1024$. Subsequently, the memory complexity is $O(\text{batch size} \times K \times C^2)$. Each FC layer is a weight tensor in $\mathbb{R}^{\text{batch size} \times z \times z}$, and its memory complexity is $O(\text{batch size} \times z^2)$. The total memory complexity of the CNN [13] is given by $O(\text{batch size} \times N_{\text{conv}} \times K \times C^2 + N_{\text{fully}} \times \text{batch size} \times z^2) = O(\text{batch size} \times (N_{\text{conv}} \times K \times C^2 + N_{\text{fully}} \times z^2)) = O(384 \times K \times C^2 + 192 \times z^2)$.

VI. CONCLUSION

In this work, the problem of masquerade detection is addressed based on variable-length UNIX command sequences. The contributions are summarized as follows.

• We introduce new features of variable-length sequences, i.e., the weighted occurrence frequencies of different orders.

• We extend the Markov-chain-based model by Xiao et al. [4] and integrate these new features into the extended model.

• For three different datasets — SEA, PU Truncated, and Greenberg Truncated, with both Shrunken and Full data settings — our model outperforms the baselines (both Markov and MarkovN2 from Xiao et al. [4]) at TPR, FPR, ROC and threshold variance metrics.

• In terms of TPR, the proposed model MarkovF2 outperformed the state-of-the-art DL model CNN [13] for PU Truncated Full and Greenberg Truncated Full, and the state-of-the-art SA-HMM [14] for SEA Full.

• Our model is much more lightweight in terms of computational and memory complexity than the state-of-the-art models. Thus, our model is more suitable for masquerade detection in real-time.

One limitation of our model is the selection of the optimal hyperparameter values. In the case of $V \leq 2$, the optimal hyperparameter space is quite small (see Table 6). In the future, we plan to test our method with higher-order ($V > 2$) weighted occurrence frequencies. In this case, because of the increased number of weight sets, which enlarges the hyperparameter space, finding optimal hyperparameters becomes more challenging. Therefore, we plan to apply a learning algorithm such as the back-propagation neural network on Markov chains proposed by Xiao et al. [38]. However, this will increase the computational cost and memory usage.

As noted in Section I-B, the state-of-the-art CNN is a text classification model applied to the masquerade detection problem. Compared to a variety of RNN- and CNN-based models, capsule neural networks (CapsNets) have demonstrated superior results in terms of accuracy and sample efficiency for a variety of NLP tasks, such as text classification [39]–[41], sentiment analysis [42], relation extraction [43], and question answering [41]. Moreover, CapsNets can be more robust against adversarial attacks like shuffling.
words, as shown in [44]. (It should be noted that the original CapsNet architecture was introduced for image classification by Sabour et al. [45].) Thus, another direction for future research is investigating the application of CapsNets to masquerade detection.

REFERENCES

[1] M. Albanese, R. F. Erbacher, S. Jajodia, C. Molinaro, F. Persia, A. Picariello, G. Sperli, and V. S. Subrahmanian, “Recognizing unexplained behavior in network traffic,” in Network Science and Cybersecurity. Berlin, Germany: Springer, 2014, pp. 39–62.

[2] R. Abdullhammed, M. Faezipour, and K. Elleithy, “Intrusion detection in delinking network: A survey,” in Intrusion Detection and Prevention for Mobile Ecosystems. New York, NY, USA: CRC Press, 2017, ch. 33, pp. 393–449.

[3] H.-S. Kim and S.-D. Cha, “Empirical evaluation of SVM-based masquerade detection using UNIX commands,” Comput. Secur., vol. 24, no. 2, pp. 160–168, Mar. 2005.

[4] X. Xiao, X. Tian, Q. Zhai, and S. Xia, “A variable-length model for masquerade detection,” J. Syst. Softw., vol. 85, no. 11, pp. 2470–2478, Nov. 2012.

[5] B. Viswanath, M. A. Bashir, M. Crovella, S. Guha, K. P. Gummadi, B. Krishnamurthy, and A. Mislove, “Towards detecting anomalous user behavior in online social networks,” in Proc. 23rd USENIX Secur. Symp. (USENIX Secur.), 2014, pp. 223–238.

[6] W. Zhu, W. Han, and X. Fu, “Online anomaly detection on e-commerce based on variable-length behavior sequence,” in Proc. 11th Int. Conf. Wireless Commun. Netw., Mobile Comput. (WiCOM), Shanghai, China, 2015, pp. 1–8, doi: 10.1049/cp.2015.0757.

[7] V. Chandola, A. Banerjee, and V. Kumar, “Anomaly detection for discrete sequences: A survey,” IEEE Trans. Knowl. Data Eng., vol. 24, no. 5, pp. 823–839, May 2012.

[8] S. K. Dash, K. S. Reddy, and A. K. Pujari, “Adaptive Naïve Bayes method for masquerade detection,” Secur. Commun. Netw., vol. 4, no. 4, pp. 410–417, Apr. 2011.

[9] Y. Vardi, M. Theusann, A. F. Karr, W.-H. Ju, W. DuMouchel, and M. Schonlau, “Computer intrusion: Detecting masquerades,” Stat. Sci., vol. 16, no. 1, pp. 58–74, Feb. 2001.

[10] T. Lane and C. E. Brodley, “An application of machine learning to anomaly detection,” in Proc. 20th Natl. Inf. Syst. Secur. Conf., Baltimore, MD, USA, vol. 377, 1997, pp. 366–380.

[11] S. Greenberg, “Using unix: Collected traces of 168 users,” Dept. Comput. Sci., Univ. Calgary, Calgary, AB, Canada, Tech. Rep. 88/333/45, 1988.

[12] R. A. Maxion, “Masquerade detection using enriched command lines,” in Proc. Int. Conf. Dependable Syst. Netw., San Francisco, CA, USA, 2003 pp. 5–14.

[13] X. Zhang and Y. LeCun, “Text understanding from scratch,” 2015, arXiv:1502.01710. [Online]. Available: http://arxiv.org/abs/1502.01710

[14] W. Ou, A. W. H. Khong, and W. P. Tay, “Hidden Markov model for masquerade detection based on sequence alignment,” in Proc. IEEE 16th Int. Conf. Dependable, Autonomic Secure Comput., 16th Int. Conf. Pervasive Intell. Comput., 4th Int. Conf. Big Data Intell. Comput. Cyber Sci. Technol. Congr. (DASC/PICOM/DataCom/CyberSciTech), Aug. 2018, pp. 278–285.

[15] K. H. Yung, “Using feedback to improve masquerade detection,” in Proc. Int. Conf. Appl. Cryptogr. Secur. Berlin, Germany: Springer, 2003, pp. 45–62.

[16] K. H. Yung, “Using self-consistent Naïve-Bayes to detect masquerades,” in Proc. Pacific-Asia Conf. Knowl. Discovery Data Mining, Berlin, Germany, 2004, pp. 329–340.

[17] A. Sharma and K. K. Paliwal, “Detecting masquerades using a combination of Naïve Bayes and weighted RBF approach.” J. Comput. Virol., vol. 3, no. 3, pp. 237–245, Jul. 2007.

[18] K. V. Reddy and N. Pushpalatha. “Conditional Naïve-Bayes to detect masquerades,” Int. J. Comput. Sci. Eng., vol. 3, no. 3, pp. 13–22, May 2014.

[19] M. Oka, Y. Oyama, H. Abe, and K. Kato, “Anomaly detection using layered networks based on Eigen co-occurrence matrix,” in Recent Advances in Intrusion Detection. Berlin: Springer, Sep. 2004, pp. 223–237.

[20] L. Chen and M. Antisugi, “An SVM-based masquerade detection method with online update using co-occurrence matrix,” in Proc. Int. Conf. Detection Intrusions Malware, Vulnerability, Berlin, Germany, 2006, pp. 57–53.
GHAZAROS BARSEGHYAN was born in Yerevan, Armenia, in 1991. He received the bachelor’s degree from the State Engineering University of Armenia and the master’s degree from the Beijing University of Posts and Telecommunications, China, where he is currently pursuing the Ph.D. degree with the School of Software. His research interests include information security, untrustworthy behavior detection, natural language processing, computer vision, machine learning, and deep learning. One of his works has been published in Mathematical Problems in Engineering.

YUYU YUAN was born in Inner Mongolia Autonomous Region, China, in 1971. She received the B.S. degree from Xidian University, in 1993, the M.S. degree from the University of Electronic Science and Technology of China, in 2000, the Ph.D. degree from the Research Institute for Fiscal Science, Ministry of Finance, China, in 2003, and the Ph.D. degree from Peking University, in 2005. She is currently a Professor and a Doctoral Supervisor with the Beijing University of Posts and Telecommunications. Her current research interests include software engineering, software quality standards, software trustworthiness, trusted service and distributed computing, blockchain technology, machine learning, smart healthcare systems, online public opinion analysis, and privacy protection. Her articles have been published in journals such as Mathematical Problems in Engineering, Journal of Systems and Software, Communications in Computer and Information Science, Applied Sciences, and Electronics.

MANAWA ANAKPA was born in Kara, Togo, in 1984. He received the B.S. and M.S. degrees in physics from the Department of Physics, Faculty of Sciences, University of Lomé, Lomé, Togo, in 2009 and 2011, respectively, and the M.S. degree in software engineering from the Beijing University of Posts and Telecommunications, Beijing, China, in 2013, where he is currently pursuing the Ph.D. degree in software engineering. His research interests include social networks, social security, personalized recommendation, electronic commerce, big data analysis, artificial intelligence, machine learning, and game theory. His work has been published in Mathematical Problems in Engineering and Communications in Computer and Information Science.