Statistical Learning for Anomaly Detection in Cloud Server Systems: A Multi-Order Markov Chain Framework

Wenyao Sha[1], Yongxin Zhu[1], Min Chen[2], Tian Huang[1]

[1] School of Microelectronics, Shanghai Jiao Tong University, China

[2] School of Computer Science and Technology, Huazhong University of Science and Technology, China

shawenyao@gmail.com, zhuyongxin@sjtu.edu.cn, minchen@ieee.org, ian_malcolm@sjtu.edu.cn

Abstract—Abstract - As a major strategy to ensure the safety of IT infrastructure, anomaly detection plays a more important role in cloud computing platform which hosts the entire applications and data. On top of the classic Markov chain model, we proposed in this paper a feasible multi-order Markov chain based framework for anomaly detection. In this approach, both the high-order Markov chain and multivariate time series are adopted to compose a scheme described in algorithms along with the training procedure in the form of statistical learning framework. To curb time and space complexity, the algorithms are designed and implemented with non-zero value table and logarithm values in initial and transition matrices. For validation, the series of system calls and the corresponding return values are extracted from classic Defense Advanced Research Projects Agency (DARPA) intrusion detection evaluation data set to form a two-dimensional test input set. The testing results show that the multi-order approach is able to produce more effective indicators: in addition to the absolute values given by an individual single-order model, the changes in ranking positions of outputs from different-order ones also correlate closely with abnormal behaviours.

Index Terms—Kth-order Markov chain, multivariate time series, anomaly detection, statistical learning

---◆---

1 INTRODUCTION

Cloud computing comes with indispensable dependency on networked computer systems. Unfortunately, while every one knows there is no guarantee of its well-being, we tend to simply ignore this painful idea. An increasing number of academical and industrial users are starting to rely solely on cloud computing servers that host entire applications and storage. In fact, these cloud computing and services in the form of distributed and open structure become obvious targets for potential threats. Thus, taking care of both business and personal data, servers expose critical safety and availability issues. Their invulnerability are of major importance to both individuals and the society.

However, during catastrophic disasters such as intrusion, crash or breakdown, the anomalies must be first discovered before any actual remedy could come to its aid. Being recessive at the early stage, such problems would not exhibit distinct traits and often lead to delayed responses and irrecoverable results. Fortunately, a server is the ideal instance whose behavior manifests regularity statistically. This lays the foundation for any anomaly detection algorithm based on machine-learning or data-mining. All of them adopt the idea of extracting the patterns within the (massive) training set and thus raise the alarm on the deviate ones.

In the literature, anomaly or intrusion detection was implemented in great variety of approaches [1]. These approaches are usually categorized into three groups, i.e. statistical approaches, machine learning approaches and data mining approaches [2], [3]:

In *statistical approaches*, anomaly or intrusion detection systems usually watch behaviors of observed objects to comprise statistical distributions as a set of trained profiles during the training phase. These systems then apply the set of trained profiles by comparing them against a new set of profiles of observed objects during the detection phase. An anomaly or intrusion is detected if these two sets of profiles do not match. In general, any incident whose occurrence frequency goes beyond standard deviations from statistical normal ranges raises an alarm [4].

Machine learning based approaches tend to reduce the supervision costs during the training phase of statistical approaches by enabling machine learning based systems to learn and improve their performance on their own. These systems are usually designed as a framework that can improve its performance in a loop cycle by adapting its execution strategies according to execution feedbacks, e.g. system call sequence analysis, Bayesian network and Markov model execution results. Neural networks and Hidden Markov Model have been proved to be useful techniques as shown in in [5] [6].

Data mining based approaches exploit unknown rules and patterns by exploring large amounts of data collected either online or off line. Anomaly or intrusion detection systems can be improved with additional inputs in terms of hidden patterns, associations, changes, and events found in data. Common technologies involved in data mining approaches include classification, clustering and outlier detection, and association rule discovery. Application of typical

data-mining algorithms such as K-nearest neighbour or clustering could be found in [7].

In the context of this paper, we focus on approaches in categories of statistical and machine learning based approaches. In [8], [9], [10], [11], theories of the classic Markov model are applied so as to detect anomalous patterns in the system, using *the ordering property of events* as proposed in [9]. [12] introduces the high-order Markov chain as an extension. Several approaches are then introduced to overcome the formidable cost that seems highly likely to come with it, including the hybrid model in [12] or support vector machine in [13].

In additional to statistical patterns, as originally proposed in [14], the time series of system calls are now by common consent a powerful tool in identifying the nature of a system's behavior. Due to system calls' privileges, a large number of researches of intrusion detection, including [15] and [16] are based on exploiting, modelling, or learning from the audit data of system calls. In 1998, the Cyber Systems and Technology Group of MIT Lincoln Laboratory conducted a 7-week simulation of intrusion in the background of daily usage, and then released all their data named as classic DARPA Intrusion Detection Evaluation Data Set [17].

The major contribution of the paper is our approach based on multi-order Markov chains, which reveals that the combination of mixed-order Markov chains would bring considerably interesting and substantial improvement over any single-order one with fairly reasonable cost. Utilizing not only the multiple order property, this approach effectively suits the application of anomaly detection in addition to its first practice in rainfall modelling in [18]. In our practice, the relative ranking positions between probabilities from multi-order models serve as a new effective indicator for anomalies, which refers to our finding that the ascending order suggests normal, while the descending one exhibits anomalous. Our approach differs from a recent model [19], which exploits mixture of Markov chains by incorporating n-gram transitions to model the normal behavior of users' HTTP requests rather than system calls in underlying servers.

Secondly, we take into accounts a new category of inputs (the return values of system calls) to improve the effectiveness of the multi-order Markov chain based approach and form a two-dimensional model. In the application of anomaly detection, the conventional notion of using system calls to identify a system's behaviour [14], [15], [16] is insufficient in that it does not take into account the resulting status of execution. In a few more recent work [20] [21], return values of system calls were to taken into consideration to detect or interpret the anomalies.

In this paper, we further looked into time spans of [17] [20] that were not explored in using our approach based on multi-order Markov chain scheme. We also provide proof to learning process as part of our scheme framework. The new time span exploration and learning process proof were not covered in [17] [20].

The remainder of the paper is organized as follows. Section 2 formulates the problem and explains the basic theory of the Markov chain model. Section 3 explains our approach to statistical training as well as our high-order models

enhanced with multivariate sequence analysis. Section 4 discusses some detailed issues involving algorithm design and complexity. A simplified example of model training and testing is also illustrated in Section 4. Validation results based on the data set is given in Section 5, and a general conclusion is presented in Section 6.

2 PROBLEM FORMULATION AND PRELIMINARIES OF MARKOV CHAIN MODEL

2.1 Problem Formulation

Fig. 1 shows a segment of typical auditing data whose first three lines indicate that the operating system opens or closes a file, reads or writes into the file and creates or terminates a process. On a much larger scale than the system calls executed, we are able to tell the difference between a normal system condition and an abnormal one as shown in the fourth row. However, on the scale as small as the first three lines, how could we learn from the case and differentiate the malicious activities? In practice, these system calls in the first three lines of Fig.1 might be used by any applications whose intentions cannot be determined as kind or malicious. To handle the uncertainty in anomaly detection due to the single indicator, results from multiple indicators or even multiple models are desired to trigger a proper alarm against prejudicial operations. In this paper, we address the above question with our scheme based on the multi-order Markov chain based model.

To make the forthcoming analysis easier to understand, it is also worth noting that concurrency is ignored on any level. Instead, it is considered as part of the deterministic nature of the auditing mechanism, which decides how every one of the simultaneous system calls gets recorded in a definitive way.

2.2 The Classical Markov Chain

In our approach, we use the discrete-time Markov chain model with finite state space to characterize the stochastic process, and it shares several remarkable similarities with the approaches for anomaly detection as already evidenced by [8], [9], [10], [11], [12], [13].

2.2.1 The Markov Property and Time-Homogeneous Assumption

Theoretically, in order for the model to hold, it is implicitly assumed that all the observation sequences satisfy the characteristics of both Markov property and time-homogeneousness, a reasonable and necessary simplification based on the potential regularity and periodicity of a server system. Consider the state space $\mathbf{X}$ with m discrete states:

$$\mathbf{X} = \{X_1, X_2, \cdots, X_m\} \tag{1}$$

and the observation sequence x_n:

$$x_n \in \mathbf{X}, \ n = 1, 2, ..., N_x \tag{2}$$

If a sequence x_n owns Markov property, then it satisfies that the present state results are solely from the last one re-

$\cdots$	10:09:10 open() success	10:09:30 ioctl() success	10:09:32 close() success	10:09:33 close() failure	10:09:34 kill() failure	10:09:40 fork() success	10:10:00 exit() success	$\cdots$
		normal				abnormal		

Fig. 1: The time series of system calls

gardless of transitions the system might have gone through before:

$$P\left(x_n = X_{i_n}|x_{n-1} = X_{i_{n-1}}, \cdots, x_2 = X_{i_2}, x_1 = X_{i_1}\right)$$
$$= P\left(x_n = X_{i_n}|x_{n-1} = X_{i_{n-1}}\right) \; for \; n \geq 2 \qquad (3)$$

In addition, the property of time-homogeneous (or stationary) ensures that the way how current state affects the next is independent of time:

$$P\left(x_{n+1} = X_j|x_n = X_i\right) = P\left(x_n = X_j|x_{n-1} = X_i\right) \quad (4)$$

Therefore, every single step of system variation would conform to a unified transition matrix, while keeping it blind to any time-sensitive anomalies.

2.2.2 The Model of Markov Chain

From the training sequence x_n in equation (2), we can derive both the initial probability distribution matrix Q, where each element q_i represents the initial probability of the corresponding state X_i:

$$Q = \left[\begin{array}{ccccc} q_1 & \cdots & q_i & \cdots & q_m \end{array} \right],$$
$$where \; q_i = P\left(x_n = X_i\right) \qquad (5)$$

and the one-step transition probability distribution matrix P, where each element p_{ij} represents the transition probability from X_i to X_j:

$$P = \left[\begin{array}{c} p_{ij} \end{array} \right]_{m \times m},$$
$$where \; p_{ij} = P\left(x_n = X_j|x_{n-1} = X_i\right). \qquad (6)$$

The model is completely specified once P and Q is given, hence we denote it as $\pi(P, Q)$.

3 STATISTICAL LEARNING FRAMEWORK, HIGH-ORDER AND MULTIVARIATE MARKOV CHAIN MODEL

3.1 Our Statistical Learning Framework

To adapt to the usual supervised learning paradigm, for the original state space $\mathbf{X}$ given in (1), we construct the first-order Markov chain transition space $\mathbf{X}^1$:

$$\mathbf{X}^1 = \{ \; [\; X_1 \; X_1 \;], \; [\; X_1 \; X_2 \;],$$
$$[\; X_1 \; X_3 \;], \; \cdots, \; [\; X_m \; X_m \;] \; \} \qquad (7)$$

which consists of every possible transition (totaling m^2) in the first-order Markov process. Similarly, we have the binary state space $\mathbf{Y}$ similar to the fourth row in Fig. 1:

$$\mathbf{Y} = \{ \; normal, \; abnormal \; \} \qquad (8)$$

So the entire set of input data $\mathbf{D}$ can be represented as coordinates mapping from space $\mathbf{X}^1$ to space $\mathbf{Y}$

$$\mathbf{D} = \left\{ (x_1^1, y_1), (x_2^1, y_2), \cdots, (x_{N_x-1}^1, y_{N_x-1}) \right\},$$
$$where \; x_n^1 \in \mathbf{X}^1 \; and \; y_n \in \mathbf{Y} \qquad (9)$$

where the series of x_n^1 is constructed using the original x_n in (2):

$$x_n^1 = [\; x_n \; x_{n+1} \;] \in \mathbf{X}^1, \; n = 1, 2, ..., N_x - 1 \qquad (10)$$

For training purpose, if series x_n (and thus x_n^1) are recorded in a completely anomaly-free environment, all y_n are set as a constant that represents normal system condition, and therefore will be considered irrelevant in the process of training.

3.1.1 Maximum Likelihood Training

In order to learn from the input data set $\mathbf{D}$, we try to find the model $\pi(P, Q)$ that maximizes the value of

$$P\left(\mathbf{D} \mid \pi(P, Q)\right) \qquad (11)$$

which is the probability of data set $\mathbf{D}$ given the model $\pi(P, Q)$ and it can be easily calculated as

$$P\left(\mathbf{D} \mid \pi(P, Q)\right) = \begin{cases} q_{x_1}, & N_x = 1 \\ q_{x_1} \prod_{n=2}^{N_x} p_{x_{n-1}x_n}, & N_x \geq 2 \end{cases} \qquad (12)$$

which equals the product of the initial probability and succeeding transition probabilities. To maximize this product, we aim to maximize every multiplier in equation (12) under the constraints of:

$$\sum_{i=1}^{m} q_i = 1 \qquad (13)$$

$$q_i \in [\, 0, 1 \,], \; i = 1, 2, \cdots, m \qquad (14)$$

$$\sum_{i=1}^{m} p_{ij} = 1, \; j = 1, 2, \cdots, m \qquad (15)$$

$$p_{ij} \in [\, 0, 1 \,], \; i = 1, 2, \cdots, m, \; j = 1, 2, \cdots, m \qquad (16)$$

From the inequality of arithmetic and geometric means, we know that once the sum of several values are predetermined, the product of these values will be maximized if and only if all of them are the same, thus:

$$q_i = \frac{The \; Number \; of \; Occurences \; of \; X_i \; in \; x_n}{N_x},$$
$$i = 1, 2, \cdots, m \qquad (17)$$

$$p_{ij} = \frac{The \; Number \; of \; Occurences \; of \; [\; X_i \; X_j \;] \; in \; x_n^1}{N_x - 1},$$
$$i = 1, 2, \cdots, m, \; j = 1, 2, \cdots, m \qquad (18)$$

Till now we have already got everything we need for the model $\pi(P, Q)$. Furthermore, we can calculate the probability of any other date set $\mathbf{D}'$ given $\pi(P, Q)$ using the same equation in (12). In a probabilistic sense, the value given in (12) also serves as a quantitative indicator of "anomaliness" by measuring how significantly the data set $\mathbf{D}'$ used in the equation deviates from the training data $\mathbf{D}$, as will be shown later.

3.2 High-order and Multivariate Markov Chain Model

As states X_i of Markov chain are intermediate derivatives, there is no specification about what kind of state X_i really is. Therefore, by constructing new states out of the original ones, we can derive a generalized variation that applies to the high-order or multivariate conditions.

3.2.1 Our High-order Markov Chain Model

Given sequence x_n that satisfies the Kth-order Markov property, where the current state depends on not only the previous statuses but also the K preceding ones:

$$
\begin{aligned}
\mathrm{P}\left(\mathrm{x_n} = \mathrm{X_{i_n}} | \mathrm{x_{n-1}} = \mathrm{X_{i_{n-1}}}, \cdots, \mathrm{x_2} = \mathrm{X_{i_2}}, \mathrm{x_1} = \mathrm{X_{i_1}}\right) \\
= \mathrm{P}\left(\mathrm{x_n} = \mathrm{X_{i_n}} | \mathrm{x_{n-1}} = \mathrm{X_{i_{n-1}}}, \cdots, \mathrm{x_{n-K}} = \mathrm{X_{i_{n-K}}}\right) \\
for \ n \geq K+1
\end{aligned}
\tag{19}
$$

we construct new sequence x_n^* from x_n:

$$
x_n^* = \underbrace{\left[\begin{array}{ccc} x_{n-K+1} & \cdots & x_n \end{array}\right]}_{K} \in \mathbf{X}^*, \ n = K, K+1, ..., N_x
\tag{20}
$$

As a sequence directly derived from x_n, x_n^* is the first-order equivalent of x_n which satisfies the classical Markov property defined in equation (3). The new state space $\mathbf{X}^*$ comprises m^K states in total:

$$
\mathbf{X}^* = \{X_1^*, \cdots, X_i^*, \cdots, X_{m^K}^*\}
\tag{21}
$$

where each state X_i^* is a certain permutation of the K original states:

$$
X_i^* = \left[\begin{array}{ccccc} X_{\theta_K(i)} & \cdots & X_{\theta_j(i)} & \cdots & X_{\theta_1(i)} \end{array}\right]
\tag{22}
$$

The permutation $\theta_j(i)$ could be defined as a base m number up to K-digits:

$$
\theta_j(i) = \frac{1}{m^{j-1}} \left\{ \left[i - \sum_{\omega=1}^{j-1}\left(\theta_\omega(i) \ m^{\omega-1}\right)\right] \ \mathrm{mod} \ m^j \right\}, \\ j = 1 \cdots K.
\tag{23}
$$

Similarly to equation (5), the initial matrix of all m^K initial states is as follows:

$$
Q^* = \left[\begin{array}{ccccc} q_1^* & \cdots & q_i^* & \cdots & q_{m^K}^* \end{array}\right]
\tag{24}
$$

where q_i^* represents the initial probability distribution of X_i^*:

$$
q_i^* = \mathrm{P}\left(x_n^* = X_i^*\right) \ for \ n \geq K.
\tag{25}
$$

Slightly different from the classical model, the total amount of possible transitions is limited to $m^K \times m$ instead of the entire $m^K \times m^K$ ones in the first-order condition, so the new transition matrix would be:

$$
P^* = \left[\begin{array}{c} p_{ij}^* \end{array}\right] \ _{m^K \times m}
\tag{26}
$$

where:

$$
p_{ij}^* = \mathrm{P}\left(x_n = X_j | x_{n-1}^* = X_i^*\right).
\tag{27}
$$

Therefore, the model in the Kth-order case is $\pi(P^*, Q^*)$.

Similar to the first-order Markov chain transition space $\mathbf{X}^1$ in (28), we construct the Kth-order Markov chain transition space $\mathbf{X}^K$:

$$
\begin{aligned}
\mathbf{X}^K = \{ \ \underbrace{[\ X_1 \ X_1 \ \cdots \ X_1 \ X_1}_{K} \ \ X_1\ ], \\
\underbrace{[\ X_1 \ X_1 \ \cdots \ X_1 \ X_2}_{K} \ \ X_1\ ], \\
\cdots, \\
\underbrace{[\ X_m \ X_m \ \cdots \ X_m \ X_m}_{K} \ \ X_m\ ]\ \}
\end{aligned}
\tag{28}
$$

which consists of every possible transition (totaling $m^K \times m$) in the Kth-order Markov process. The label space $\mathbf{Y}$ remains unchanged. Now input data set $\mathbf{D}$ can also be represented as coordinates mapping from space $\mathbf{X}^K$ to space $\mathbf{Y}$

$$
\begin{aligned}
\mathbf{D} = \left\{(x_1^K, y_1), (x_2^K, y_2), \cdots, (x_{N_x-K}^K, y_{N_x-K})\right\}, \\
where \ x_n^K \in \mathbf{X}^K \ and \ y_n \in \mathbf{Y}
\end{aligned}
\tag{29}
$$

where the series of x_n^K is constructed using the original x_n in (2) and x_n^* in (20):

$$
x_n^K = \left[\ x_n^* \ x_{n+1}\ \right] \in \mathbf{X}^K, \ n = K, K+1, ..., N_x - 1
\tag{30}
$$

Correspondingly, equation (12) becomes:

$$
\mathrm{P}\left(\mathbf{D} \mid \pi(P^*, Q^*)\right) = \begin{cases} q_{x_K^*}^*, & N_x = K \\ q_{x_K^*}^* \prod\limits_{n=K+1}^{N_x} p_{x_{n-1}^* x_n^*}^*, & N_x \geq K+1 \end{cases}
\tag{31}
$$

3.2.2 Enhancement with Multivariate Sequences

Given multiple sequences for anomaly detection, it is intuitive to apply the model to each of them in separation. In this way, the hidden correlation among multiple variables in these sequences are lost unfortunately, which might be the very factor that contributes to anomalies. For example, consider the following case, where Fig. 2 is a piece of

10:09:20	10:09:21	10:09:22	10:09:24	10:09:26
fork()	fork()	kill()	open()	open()
success	success	failure	success	failure

Fig. 2: Normal Series

10:09:20	10:09:21	10:09:22	10:09:24	10:09:26
fork()	fork()	kill()	open()	open()
success	success	failure	success	success

Fig. 3: Test Series 1

10:09:20	10:09:21	10:09:22	10:09:24	10:09:26
fork()	fork()	kill()	open()	open()
failure	failure	success	failure	sucess

Fig. 4: Test Series 2

time series collected in anomaly-free context, and we are evaluating the likelihood of the test series in Fig. 3 and Fig. 4 being also normal.

From a stand-alone point of view, all the transitions of system calls and return values are legitimate, as their occurrences have already been observed in the training series. The only difference is that Fig. 3, being almost the same as the training series, may yield a higher likelihood than Fig. 4. However, a closer look at the joint states of system calls and return values tells us that none of the transitions in Fig. 4 are possible. For example, the fork command with the return value of failure can never transit into another fork with the return value of failure in the normal series, which means Fig. 4 actually stands for something more extreme than the transitions along each dimension suggests. In other words, looking at only one dimension of system calls or return values at a time gives us less information than looking at them jointly.

Therefore, combing these variables into a new space would be a practically superior enhancement. Combining multiple inter-related sequences as a multivariate into a single model would be another feasible approach to improve the sensitivity of detection. The example shown in Fig. 2, Fig. 3 and Fig. 4 is not detectable by major existing methods in the form of statistical pattern detection and time series analysis [8] [9] [10] and [15].

For a S-dimensional sequence, if the original state spaces are noted as $\mathbf{X}_1$, $\mathbf{X}_2$ to $\mathbf{X}_S$ and each consists of m_1, m_2 or m_S states respectively, the corresponding state space of combination comprises totaly m^S states:

$$m^S = \prod_{i=1}^{S} m_i \tag{32}$$

and the new state space $\mathbf{X}(S)$ is given by:

$$\mathbf{X}(S) = \{X_1(S), \cdots, X_i(S), \cdots, X_{m^S}(S)\} \tag{33}$$

each state $X_i(S)$ of which is a combination of the original states:

$$X_i(S) = \begin{bmatrix} X_{i_1} & \cdots & X_{i_j} & \cdots & X_{i_S} \end{bmatrix},$$
$$where \ 1 \leq i_j \leq m_j \ and \ j = 1, 2, ..., S \tag{34}$$

where each element belongs to one of the original state spaces:

$$X_{i_j} \in \mathbf{X}_j, \ j = 1, 2, ..., S \tag{35}$$

Hence, the multivariate sequences are able to comply with the classical univariate Markov chain model $\pi(P, Q)$ described in previous sections.

Moreover, the variation of high-order and multivariate model could be applied jointly, creating a S-dimensional Kth-order Markov chain model $\pi[P^*(S), Q^*(S)]$ with S-dimensional Kth-order Markov chain transition space $\mathbf{X}^K(S)$.

Though combination of sequences often results in higher time and space complexity, in our practice of algorithm design discussed later, both of these generalizations are implemented with complexity reduction techniques as the Multi-order Markov Chain based model, where we utilize the system calls and their return values as a two-dimensional input ($S = 2$) of the 1st, 2nd and 3rd-order Markov chain ($K = 1, 2, 3$).

4 ALGORITHM DESIGN AND IMPLEMENTATION

To formulate our algorithm, we follow the process of anomaly detection workflow comprising of training stage and test stage. On top of the generic training and testing stages, we further tackle the issues of model order selection, zero-probability event transform and sparse matrix storage etc. Our solutions to these issues will be explained before the complete algorithm is formulated.

4.1 Design Issues

Order of the model in our approach is the trade-off between model complexity and generalizability. System of higher-order tend to neutralize the idealization brought by the assumption of Markov property and thus provides a more sensitive response over anomalous exceptions. In other words, high-order Markov chain generates larger support for frequent patterns within the training set while producing smaller support for rare ones at the same time. On the contrary, high-order models might lead to overfitting for noise or idiosyncrasy of the training sequence, and loses the ability to generalize outside it. In consequence, Markov chain models of up to the third order are utilized in our simulation.

Zero-probability-event issues refer to the fact that due to the non-exhaustive nature of any training sequence, one or more elements in P or Q will remain zero after training, and a single zero in the multipliers would ultimately dominate the whole output value. Here a solution proposed in [8] is implemented by replacing the zeros with a value far smaller than any other element (yet still essentially different from the real zero).

Another typical issue resulting from the non-exhaustiveness of training set is the occurrence of completely new states in test sets. Even after going through all the training data, what we consider as the state space $\mathbf{X}$ could just be a subset of the real one because some states simply might not have occurred at all in the recording of training set. So for application purpose, we add a new artificial state X_{m+1} to the original space $\mathbf{X}$ to represent "everything else". In this way, without compromising any accuracy of calculation, state space $\mathbf{X}$ will be independent of the test set and we will not have to deal with an growing $\mathbf{X}$ as we proceed.

4.2 Algorithm Design

The final algorithms that take both zero-probability events and storage optimization into consideration are presented as Algorithm 1-4. Algorithm 1, *Training*, represents the training stage which processes the training set and generates the initial probability distribution and transition probability distribution matrices, of which the latter one is the major result of the training stage. Algorithm 2, *Testing*, takes the two matrices and the test set in and calculates its probability of occurrence given the model trained. Algorithm 1 and algorithm 2 also rely on two functions i.e. *IncreaseTransitionMatrix* and *GetTransitionMatrix* to calculate transition probabilities and retrieve values in these matrices.

Algorithm 3 and algorithm 4 are two functions called during training and testing, which either increases a specific element of transition matrix by one or returns its value.

Algorithm 1 Training()

Input: Training sequence $x_n \in \mathbf{X}$ ($n = 1 \to N_x$), Number of states m, Order K
Output: Initial probability distribution matrix Q, Transition probability distribution matrix P
  1: Initialize()   // Set Q and P equal to 0
  2: **for** $n = K \to N_x$ **do**
  3:     $x_n^* \leftarrow \begin{bmatrix} x_{n-K+1} & x_{n-K+2} & x_{n-K+3} & \cdots & x_n \end{bmatrix}$ // Construct new sequence in space $\mathbf{X}^*$
  4: **end for**
  5: **for** $n = K \to N_x - 1$ **do**
  6:     $x_n^K \leftarrow \begin{bmatrix} x_n^* & x_{n+1} \end{bmatrix}$ // Construct new sequence in space $\mathbf{X}^K$
  7: **end for**
  8: **for** $n = K \to N_x$ **do**
  9:     **if** $x_n = X_i$ **then**
 10:         IncreaseInitialMatrix(i)   // Get the index of x_n and increase q_i^* in Q by 1
 11:     **end if**
 12: **end for**
 13: **for** $n = K \to N_X - 1$ **do**
 14:     **if** $x_n^K = \begin{bmatrix} X_i^* & X_j \end{bmatrix}$ **then**
 15:         IncreaseTransitionMatrix(i, j)   // Get the index of x_n^K and increase p_{ij}^* in P by 1
 16:     **end if**
 17: **end for**
 18: NormalizeInitialMatrix()   // Divide the matrix Q by $N_x - K$
 19: NormalizeTransitionMatrix()   // Divide the matrix P by $N_x - K - 1$

Algorithm 2 Testing()

Input: Testing sequence $y_n \in \mathbf{X}$ ($n = 1 \to N_y$), Q, P
Output: Probability of y_n given Q and P
  1: **for** $n = K \to N_y$ **do**
  2:     $y_n^* \leftarrow \begin{bmatrix} y_{n-K+1} & y_{n-K+2} & y_{n-K+3} & \cdots & y_n \end{bmatrix}$ // Construct new sequence in space $\mathbf{X}^*$
  3: **end for**
  4: **for** $n = K \to N_y - 1$ **do**
  5:     $y_n^K \leftarrow \begin{bmatrix} y_n^* & y_{n+1} \end{bmatrix}$ // Construct new sequence in space $\mathbf{X}^K$
  6: **end for**
  7: **for** $n = K \to N_y$ **do**
  8:     **if** $y_n = X_i$ **then**
  9:         probability $\leftarrow$ GetInitialMatrix(i)   // Get q_i^* in Q
 10:     **end if**
 11: **end for**
 12: **for** $n = K \to N_y - 1$ **do**
 13:     **if** $y_n^K = \begin{bmatrix} X_i^* & X_j \end{bmatrix}$ **then**
 14:         probability $\leftarrow$ probability $\times$ GetTransitionMatrix(i, j)   // Multiply by p_{ij}^* in P
 15:     **end if**
 16: **end for**
 17: Output : probability

4.3 Time and Space Complexity

The training stage involves calculating the initial and transition matrices and takes approximately

$$O\left(m^{K+1}\right) \tag{36}$$

's time, which grows either polynomially with the amount of states or exponentially with the order of Markov chain. The test stage requires nothing more than a series of multiplication and consumes about

$$O\left(N_x\right) \tag{37}$$

's time, which is linearly proportional to the length of the testing data set.

On the other hand, the algorithm requires the matrix P^* and Q^* be accessible at all time during the training and testing stage, so it can consume memory space as large as

$$O\left(m^{K+1}\right) \tag{38}$$

4.4 Curbing the Complexity

The training algorithm, acting as a forefront overhead for the testing algorithm, does not affect the real-time performance of anomaly detection. Also, the testing algorithm is simply a serial of continual multiplication which will hardly become the burden of any modern processor. Even in the cases where multiplying are considered unacceptably time-consuming, it would be highly feasible to replace both transition and initial matrices with their logarithm values

---

Algorithm 3 IncreaseTransitionMatrix(i, j)

Input: The index of state in transition(i and j)
Output: Transition matrix (stored in boolean values), Nonzero value table

1: $MAXNONZERO \leftarrow 1 \times 10^4$ // Define max number of nonzero values
2: **if** TransitionMatrixBit[i][j] = 0 **then**   // If the nonzero value does not exist
3:   CounterOfNonzeroValues $\leftarrow$ CounterOfNonzeroValues + 1   // Increase counter
4:   **if** CounterOfNonzeroValues > MAXNONZERO **then**   // If exceeded
5:     Print : Error! Please increase MAXNONZERO!   // Print message
6:     exit   // Algorithm aborted
7:   **end if**
8:   iOfNonzeroValues[CounterOfNonzeroValues] $\leftarrow$ i   // Set index of nonzero value
9:   jOfNonzeroValues[CounterOfNonzeroValues] $\leftarrow$ j   // Set index of nonzero value
10:   NonzeroValues[CounterOfNonzeroValues] $\leftarrow$ 1   // Set initial value
11:   TransitionMatrixBit[i][j] $\leftarrow$ 1   // Set sign of the nonzero value
12: **else**   // If the nonzero value already exists
13:   **for** $n = 1 \rightarrow$ CounterOfNonzeroValues **do**
14:     **if** iOfNonzeroValues[n] = i AND jOfNonzeroValues[n] = j **then**
15:       break   // Find where the nonzero value is stored
16:     **end if**
17:   **end for**
18:   NonzeroValues[n] $\leftarrow$ NonzeroValues[n] + 1   // Increase the value by 1
19: **end if**

Algorithm 4 GetTransitionMatrix(i, j)

Input: The index of state in transition(i and j), Nonzero value table, Transition matrix (stored in binary bits)
Output: Probability of the transition from state i to state j

1: $ZERO \leftarrow 1 \times 10^{-5}$
2: **if** TransitionMatrixBit[i][j] = 0 **then**   // If it is a zero
3:
4:   **return** ZERO
5: **else**   // If it is a nonzero value
6:   **for** $n = 1 \rightarrow$ CounterOfNonzeroValues **do**
7:     **if** iOfNonzeroValues[n] = i AND jOfNonzeroValues[n] = j **then**
8:
9:       **return** NonzeroValues[n]   // Find where the nonzero value is stored
10:     **end if**
11:   **end for**
12: **end if**

---

in the training stage beforehand, causing all multiplication of the test stage substituted by addition, one of the basic commands supported by any instruction set architecture.

As the amount of states and the order of model grow, the memory consumption for storing transition matrix alone could be formidable. However, when taking into account the sparse feature of the matrix (which assumes a significant amount of potentially possible transitions does not occur in reality), a binary bit of memory space is already sufficient for most of its elements. Together with the extra table declared exclusively for saving all nonzero values and their indices (with subjectively pre-determined array size), the revised algorithm nonetheless succeeds in distinctly optimizing space complexity at the cost of limited sacrifice in time complexity. In our practice, the length of the non-zero value table is set to 10^4 and no overflow has occurred, another proof of how such a look-up approach manages to save storage consumption without compromising any accuracy.

4.5 Implementation Illustration with an Example

For clarity, an example of model training and testing is shown step by step as follows. Suppose the training data set **D**, which are the series of system calls and return values from a two-virtual-machine system in its normal running state, are shown in Fig. 5 and Fig. 6, and a two-dimensional ($S = 2$) model of 2nd-order ($K = 2$) is desired.

10:09:20	10:09:21	10:09:22	10:09:24	10:09:26
kill()	fork()	kill()	fork()	open()
success	failure	success	failure	failure
		normal		

Fig. 5: System call series of Virtual Machine 1 (training set)

First, the two-state space A and three-state space B could be obtained and noted as:

$$A = \{success, \ failure\} = \{a_1, \ a_2\} \qquad (39)$$

$$B = \{open(), \ kill(), \ fork()\} = \{b_1, \ b_2, \ b_3\} \qquad (40)$$

10:09:20	10:09:21	10:09:22	10:09:24	10:09:26
fork()	fork()	kill()	open()	open()
failure	failure	success	failure	success
		normal		

Fig. 6: System call series of Virtual Machine 2 (training set)

For simplicity, the training sequences of return values and system calls derived directly from Fig. 5 and 6 are noted as RV_1, SC_1, RV_2 and SC_2 respectively:

$$RV_1 = a_1, a_2, a_1, a_2, a_2 \qquad (41)$$

$$SC_1 = b_2, b_3, b_2, b_3, b_1 \qquad (42)$$

$$RV_2 = a_2, a_2, a_1, a_2, a_1 \qquad (43)$$

$$SC_2 = b_3, b_3, b_2, b_1, b_1 \qquad (44)$$

Now we create the table for mapping from the input multivariate sequence to a univariate one as shown in TABLE 1, and construct a univariate equivalent sequence x_n (and the

TABLE 1: Mapping from $[\ RV_1\ SC_1\ RV_2\ SC_2\ ]$ to $\mathbf{X}$

$[\ RV_1\ SC_1\ RV_2\ SC_2\ ]$	$\mathbf{X}$
$[\ a_1\ b_1\ a_1\ b_1\ ]$	X_1
$[\ a_1\ b_1\ a_1\ b_2\ ]$	X_2
$[\ a_1\ b_1\ a_1\ b_3\ ]$	X_3
$\cdots$	$\cdots$
$[\ a_2\ b_3\ a_2\ b_1\ ]$	X_{34}
$[\ a_2\ b_3\ a_2\ b_2\ ]$	X_{35}
$[\ a_2\ b_3\ a_2\ b_3\}$	X_{36}

implicit state space $\mathbf{X}$):

$$x_n = \{a_1, b_2, a_2, b_3\}, \{a_2, b_3, a_2, b_3\},$$
$$\{a_1, b_2, a_1, b_2\}, \{a_2, b_3, a_2, b_1\},$$
$$\{a_2, b_1, a_1, b_1\} = X_{12}, X_{36}, X_8, X_{34}, X_{19} \qquad (45)$$

TABLE 2: Mapping from $[\ x_n\ x_n\ ]$ to $\mathbf{X}^*$

$[\ x_n\ x_n\ ]$	$\mathbf{X}^*$
$[\ X_1\ X_1\ ]$	X_1^*
$[\ X_1\ X_2\ ]$	X_2^*
$[\ X_1\ X_3\ ]$	X_3^*
$\cdots$	$\cdots$
$[\ X_{36}\ X_{34}\ ]$	X_{1294}^*
$[\ X_{36}\ X_{35}\ ]$	X_{1295}^*
$[\ X_{36}\ X_{36}\ ]$	X_{1296}^*

Then, the first-order equivalent sequence x_n^* (and the implicit state space $\mathbf{X}^*$) is given using the mapping in TABLE 2:

$$x_n^* = \{X_{12}, X_{36}\}, \{X_{36}, X_8\}, \{X_8, X_{34}\}, \{X_{34}, X_{19}\}$$
$$= X_{432}^*, X_{1268}^*, X_{286}^*, X_{1207}^* \qquad (46)$$

So,

$$x_n^K = [\ X_{432}^*, X_8\ ], [\ X_{1268}^*, X_{34}\ ], [\ X_{286}^*, X_{19}\ ] \qquad (47)$$

Evidently, the initial matrix Q^* (1 by 6×6) with only four nonzero elements ($q_{286}^* = q_{432}^* = q_{1207}^* = q_{1268}^* = 1/4$):

$$Q^* = \left[\ \cdots\ \ \tfrac{1}{4}\ \ \cdots\ \ \tfrac{1}{4}\ \ \cdots\ \ \tfrac{1}{4}\ \ \cdots\ \ \tfrac{1}{4}\ \ \cdots\ \right]_{1 \times 1296} \qquad (48)$$

and the transition matrix P^* (36×36 by 36) with only three nonzero elements ($p_{432,8}^* = p_{1268,34}^* = p_{286,19}^* = 1$):

$$P^* = \begin{bmatrix} \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \cdots & 1 & \cdots & \cdots & \cdots & \cdots & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \cdots & \cdots & \cdots & 1 & \cdots & \cdots & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & 1 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix}_{1296 \times 36} \qquad (49)$$

are established, which means we have the model $\pi(P^*, Q^*)$. Therefore, the probability of every test sequence constructed this way could be determined using equation (31). For instance, given newly-observed series $\mathbf{D}'$ shown in Fig. 7 and Fig. 8 as the test input sets, again the test sequences

10:19:30	10:19:31	10:19:32
fork()	kill()	fork()
failure	success	failure

Fig. 7: System call series of Virtual Machine 1 (test set 1)

10:19:30	10:19:31	10:09:32
fork()	kill()	open()
failure	success	failure

Fig. 8: System call series of Virtual Machine 2 (test set 1)

derived are noted as RV_1', SC_2', RV_1', SC_2':

$$RV_1' = a_2, a_1, a_2 \qquad (50)$$

$$SC_1' = b_3, b_2, b_3 \qquad (51)$$

$$RV_2' = a_2, a_1, a_2 \qquad (52)$$

$$SC_2' = b_3, b_2, b_1 \qquad (53)$$

Similarly, we have:

$$x_n' = \{a_2, b_3, a_2, b_3\}, \{a_1, b_2, a_1, b_2\}, \{a_2, b_3, a_2, b_1\}$$
$$= X_{36}, X_8, X_{34} \qquad (54)$$

$$x_n'^* = \{X_{36}, X_8\}, \{X_8, X_{34}\} = X_{1268}^*, X_{286}^* \qquad (55)$$

$$x_n'^K = [\ X_{1268}^*, X_{34}\ ] \qquad (56)$$

According to equation (31), the probabilities of series X'^* given the model $\pi(P^*, Q^*)$ are:

$$\mathrm{P}\left(\mathbf{D}'\mid \pi(P^*, Q^*)\right) = q_{1268}^* \times p_{1268,34}^* = 1 \times 1 = 1 \qquad (57)$$

On the other hand, for newly-observed series shown in Fig. 9 and Fig. 10, by applying the same methodology as test set 1 and 2, their probability of occurrence given the model $\pi(P^*, Q^*)$ equals 0. As a result, the latter test set is expected to be more "abnormal" than the former one since the latter test set has a lower probability of support than the former

10:59:30	10:59:31	10:59:32
open()	fork()	kill()
success	success	failure

Fig. 9: System call series of Virtual Machine 1 (test set 2)

10:59:30	10:59:31	10:59:32
kill()	fork()	open()
failure	success	success

Fig. 10: System call series of Virtual Machine 2 (test set 2)

one does, which means decision can be made according to the probability of support.

In addition, the ordinal indices of the new state space could conform to any mapping other than TABLE 1 and TABLE 2 as long as it maintains uniqueness.

5 VALIDATION RESULTS AND DISCUSSION

As a real-world application of the multi-order Markov chain anomaly detecting framework, we take in the classic DARPA Intrusion Detection Evaluation Data Set [17] by the Cyber Systems and Technology Group of MIT Lincoln Laboratory to verify our scheme. The DARPA dataset were collected since June 1998 when DARPA conducted a 7-week simulation of TCP attacks (anomalies) in the background stream of normal and regular user operations. Major results for comparison [8][9][10] and [15] were achieved with this data set. The rest data sets in relevant literature were either unavailable to the public or even older than DARPA data set. Hence, we just choose DARPA data set. In [11], authors conducted their experiments on genomic data rather than security data sets. In [12], data were collected from the output of the UNIX acct auditing mechanism on a local machine at authors lab, which were not available to the public. In [16] published in 2006, an even earlier data set (CERT synthetic Sendmail data collected at the University of New Mexico (UNM) by Forrest et al. (1996)) was used. In [19], an unpublished data set on authors own web servers was used, which was not available for cross verification.

With the help of Basic Security Module (BSM) provided by Oracle Solaris, system information including event time, types of system call executed, the return value, et al. were audited during the experiment. A typical piece of BSM audit data can be found in Fig. 11.

5.1 Validation Results Regarding Multi-order Markov Chain Model

For training set $\mathbf{D}$, we apply algorithm 1, *Training*, on all the 819472 data of system call sequence recorded on the first week's Friday, while data from other Fridays are used as input for algorithm 2, *Testing*, respectively. A more detailed summary of input parameters and execution could be found in TABLE 3 and TABLE 4. It is also noteworthy that although BSM is able to monitor up to 243 types of system calls, only 57 of them were encountered during the training stage. In this case, treating all other unencountered system calls as a single category would result in a simplified yet equivalent state space comprising totally 58 states.

TABLE 3: Input Parameters of Whole Day

Parameter	Value
Orders	1,2,3
Training set amount	819472
Test set amount	100000×6
Sliding window width	200
ZERO	1×10^{-5}

TABLE 4: Multi-order Execution Summary for Data of Whole Day

Order	1	2	3
Number of states	58	3364	195112
Number of transitions	3364	195112	11316496
Typical runtime (training)	3s	3s	5s
Typical runtime (test)	25s	38s	56s

It is worth noting that the data inputs in TABLE 3 and TABLE 4 covered 24 hours of the first week's Friday. This amount of data only takes up to 56 seconds to complete our testing detection. If we feed a segment of data covering only half an hour into our detection scheme for testing, it is expected to take around 1 second to complete the testing. The reason behind the quick response in our detection scheme is that kernel calculations involved in the testing phase only include looking up for the probability of the new states and multiplying sets of probabilities together. In other words, our scheme is able to be deployed for on-line anomaly detection.

Calculation results of week 2 to 7 using the multi-order Markov chain model are shown in Fig. 12, where the horizontal axis represents the ordinal number of a subset of testing sequence with 200 data (a sliding window over the series with a width of 200), while the vertical axis represents the negative logarithmic values of the probability of the subset of test series $\mathbf{D}_{testing}$ given the model $\pi(P^*, Q^*)$ just trained:

$$-log\left\{ \mathrm{P}[\, \mathbf{D}_{testing} \mid \pi(P^*, Q^*)\,]\right\} \tag{58}$$

The blue, red and green curves stand for the first, second and third-order model respectively.

As many figures show, for most of the time, the calculation results are maintained at a relatively low level, following the descending order from the first-order model to higher ones. Take the third-order model in Fig. 12 (a) for instance, the green curve usually oscillates between 20 and 80, which indicates an average one-step transition probability of approximately 0.4 to 0.8 due to the inverse function of negative logarithm as shown in

$$(10^{-20})^{1/200} \approx 0.80 \tag{59}$$

$$(10^{-80})^{1/200} \approx 0.40 \tag{60}$$

In view of the total 58 possible transitions, a probability of 0.4 to 0.8 would by no means be regarded as a "rare" event and thus agrees with the universal rule of the normality having a greater probability. However, at some point, the curve rises sharply, reaching roughly 330, or

$$(10^{-330})^{1/200} \approx 0.02 \tag{61}$$

in terms of average one-step transition probability somewhere in the tail of Fig. 12 (a), which is significantly smaller

...

```
header,150,2,ioctl(2),,Fri Jun 05 07:49:06 1998, + 499467935 msec
path,/etc/security/audit control
attribute,100664,root,other,8388608,62781,0
argument,2,0x5401,cmd
argument,3,0xefffed7c,arg
subject,2122,root,other,root,other,257,257,0 0 pascal.eyrie.af.mil
return,failure: Inappropriate ioctl for device,-1
trailer,150
```

...

Fig. 11: A typical segment of BSM audit data

TABLE 5: Multivariate Input Parameters

Parameter	Value
Orders	1,2,3
Dimension	2
Training set amount	819472×2
Test set amount	$(100000 \times 2) \times 6$
Sliding window width	200
ZERO	1×10^{-5}

TABLE 6: Multivariate Execution Summary

Order	1	2	3
Number of states	116	3364	1560896
Number of transitions	13456	1560896	181063936
Typical runtime (conversion)		15s	
Typical runtime (training)	3s	4s	5s
Typical runtime (test)	27s	44s	68s

than any one-step transition probability encountered before. At the same time of the "spike", an abrupt reverse of the originally descending order from the first to higher-order model has also occurred. Moreover, the sixth Friday shown in Fig. 12 (e) witnesses a much more dramatic situation in which the third-order result soars to 1000, or an average one-step transition probability of

$$(10^{-1000})^{1/200} = 10^{-5} \qquad (62)$$

which is the exact same value as the pre-defined constant ZERO in TABLE 3. This means that for these particular subsets of the testing set, every single transition in them has gone from "rare" or "low probability" to "impossible" or "never happened before".

According to the record by Lincoln Lab, both of the extreme values depicted in Fig. 12 (a) and Fig. 12 (e) coincide with a TCP attack chronologically, whereas the latter one resulted in an unprecedented system crash on July 10th, 1998.

5.2 Validation Results Regarding Multivariate Sequence

Whenever a system call is executed, a certain binary value of either SUCCESS or FAILURE is returned, which can also be obtained from BSM as shown in Fig. 11. Along with the types of system calls processed above, the return values contribute to forming a kind of simplest two-dimensional time series in a server system. Accordingly, the summary of input parameters and execution can be found in TABLE 5 and TABLE 6.

The multivariate computing results of the same Fridays in previous figures are shown in Fig. 13. Now the vertical axis has a slight new meaning of the negative logarithmic values of the probability of the subset of test series $\mathbf{D}_{testing}$ given the multi-variate model $\pi[P^*(S), Q^*(S)]$:

$$-log \{ P[\mathbf{D}_{testing} \mid \pi(P^*(S), Q^*(S))] \} \qquad (63)$$

instead of equation (58).

Although somewhat similar at first glance, most of these two sets of results could be easily distinguished in many aspects, e.g., where the univariate model in Fig. 12 (a) gives 330 to indicate anomalies, the multivariate one generates 400 in Fig. 13 (a), which further reduces the average one-step transition probability of 0.02 in equation (61) to

$$(10^{-400})^{1/200} = 0.01 \qquad (64)$$

The greater values from the multivariate model $\pi[P^*(S), Q^*(S)]$ (or smaller values in terms of average one-step transition probability) naturally lead to a more sensitive anomaly detection considering the fact that when intrusion or attack occurs, it is more likely to have an extensive scale of system calls executed with the return value of FAILURE.

Other multivariate results like the sixth Friday in Fig. 13 (e) fail to yield a better detection, as the curves in Fig. 12 (e) are already at its theoretical maximum of 1000 (or 10^{-5} as average one-step transition probability) using the univariate approach.

5.3 Discussion

Essentially, the high-order Markov chain model $\pi(P^*, Q^*)$ comes with far more degrees of freedom than the first-order model $\pi(P, Q)$. Therefore, after sufficient training, $\pi(P^*, Q^*)$ is expected to extract "more" information from the training set regarding normal transition patterns, but at the cost of fitting some idiosyncratic noises as well. In the application above, the high-order model $\pi(P^*, Q^*)$ acts like a signal amplifier that increases the level of "normalness" (or "abnormalness") to a normal (or abnormal) transition. It is this trait that results in the distinct inversion of relative positions from the multi-order models when anomalies occur, and this relation can be useful in detecting anomalies along with the conditional probabilities themselves given by different-order models.

(a) Friday, Week 2

(b) Friday, Week 3

(c) Friday, Week 4

(d) Friday, Week 5

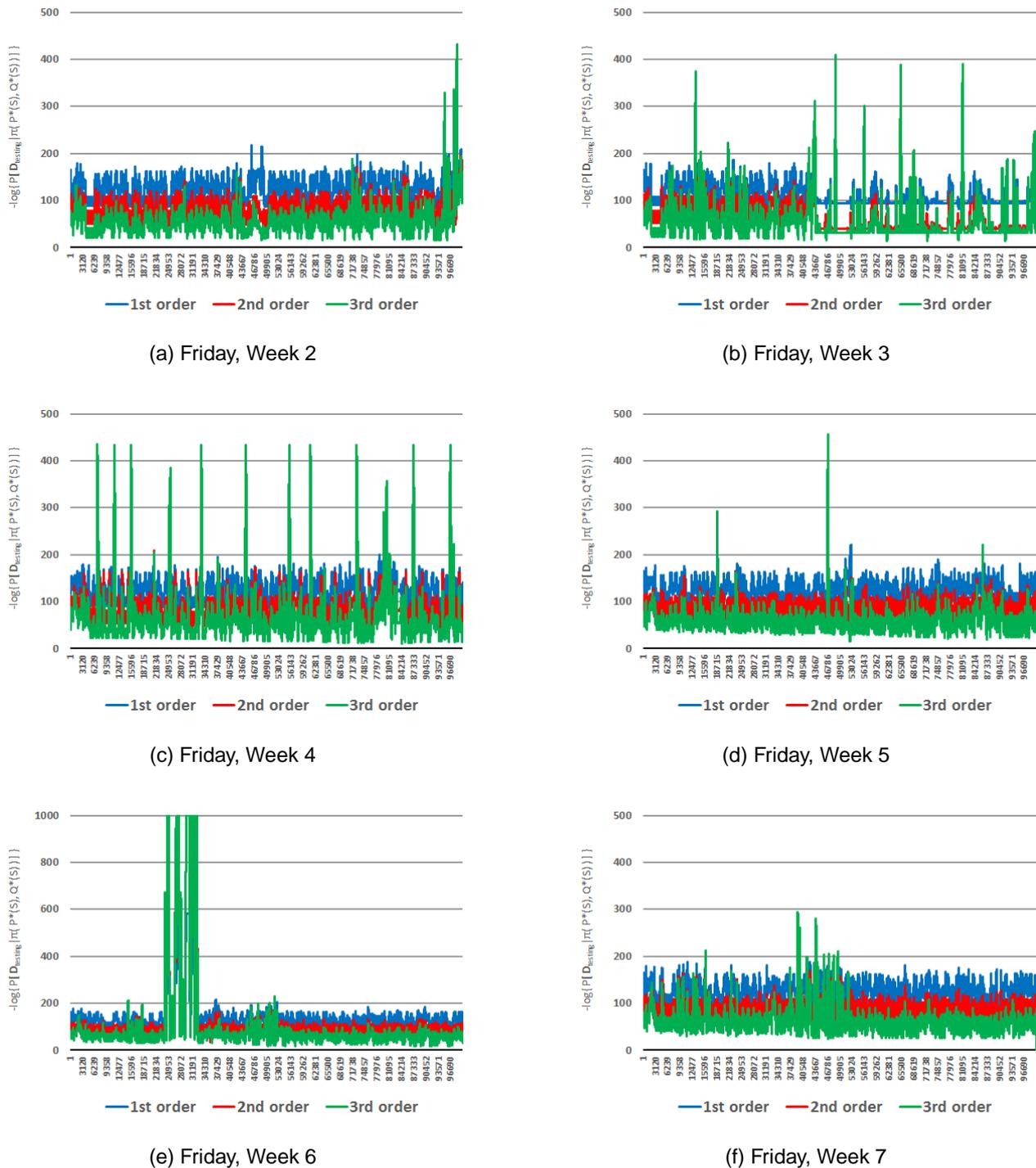(e) Friday, Week 6

(f) Friday, Week 7

Fig. 12: Calculation results from Friday, Week 3 to 7, where the horizontal axis represents the ordinal number of a sub training sequence with 200 data, and the vertical axis represents the negative logarithmic values of support from the model using the first Friday as the training input. The blue, red and green curves stand for the first, second and third-order model respectively. In other words, the specific steps of generating these results are: 1). Input the data from Friday, Week 1 into Algorithm 1, and by counting the occurrence of every possible state and transition, set up the initial and transition matrix; 2). Input the data from other Fridays into Algorithm 2, and multiply the corresponding elements of the former two matrices to get the conditional probability of the test set given the model; 3). For convenience, the negative logarithmic value are taken as output. Note that all following figures are presented in the same way.

However, it is still perfectly possible to come up with a certain training set that leads to a model with completely opposite properties, which inevitably requires a cautious selection for the training set. After all, it is what is written in the training data that defines the normal and the anomaly in any application. In our study, the unique settings of a server

(a) Friday, Week 2 (two-dimensional)

(b) Friday, Week 3 (two-dimensional)

(c) Friday, Week 4 (two-dimensional)

(d) Friday, Week 5 (two-dimensional)

(e) Friday, Week 6 (two-dimensional)

(f) Friday, Week 7 (two-dimensional)
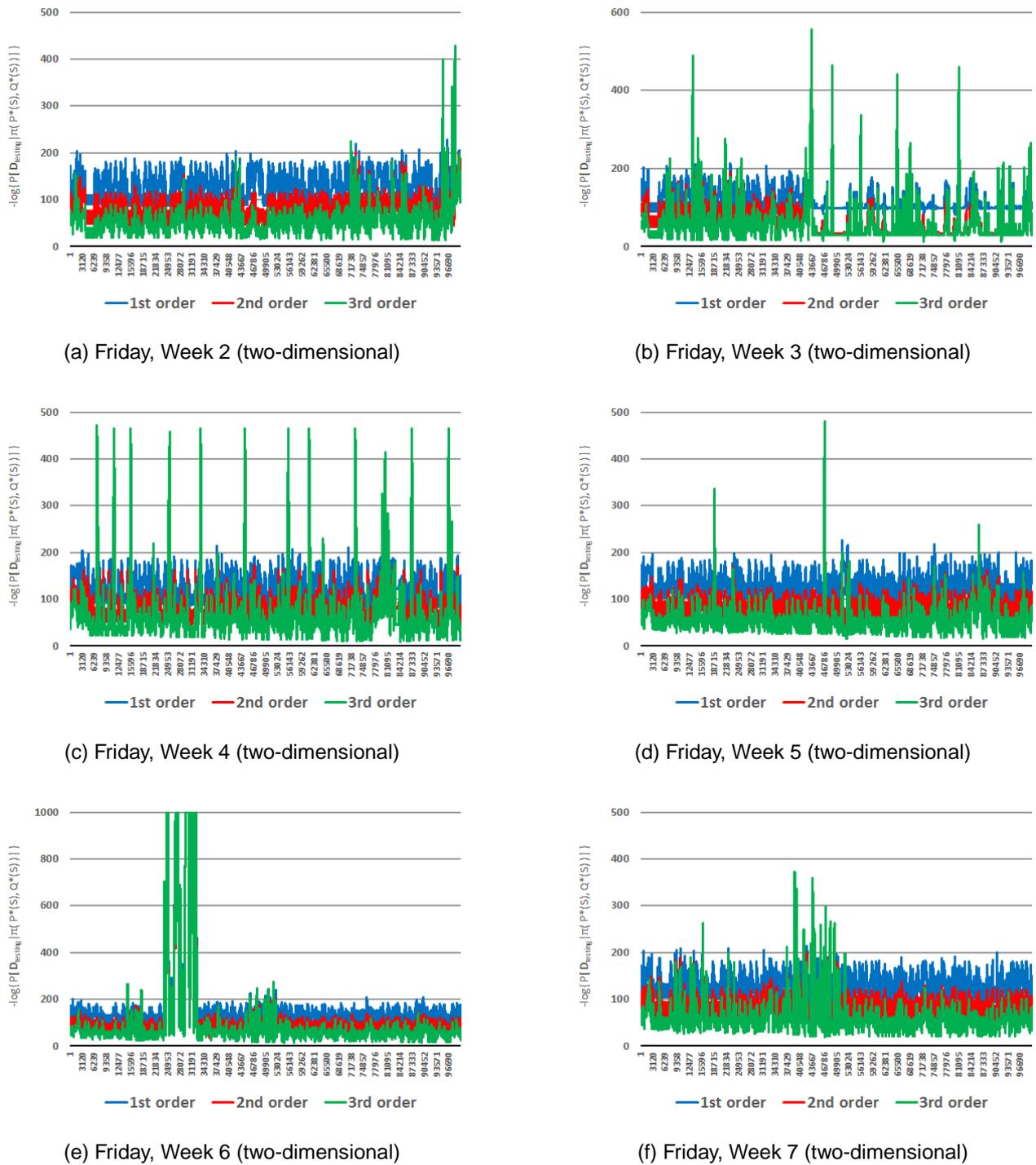
Fig. 13: Friday, Week 3 to 7 (two-dimensional)

system helps to relax such constraints to some extent.

To summarize, using the *Training* algorithm and *Testing* algorithm, we provide the following rules of thumb:

1) Relations between $\mathrm{P}[\mathbf{D}_{testing}|\pi(P,Q)]$ and $\mathrm{P}[\mathbf{D}_{normal}|\pi(P^*,Q^*)]$ can be regarded a sign of anomaly. For normal (or common) testing data set $\mathbf{D}_{normal}$, while the first-order model $\pi(P,Q)$ will already assign relatively large value to $\mathrm{P}[\mathbf{D}_{normal}|\pi(P,Q)]$, higher-order model $\pi(P^*,Q^*)$ tends to give even larger value to the probability

of the corresponding higher-order transitions in space $\mathbf{X}^K$. On the other hand, for abnormal (or rare) testing data set $\mathbf{D}_{anomaly}$, while the first-order model $\pi(P,Q)$ will already assign relatively small (or ZERO) value to $\mathrm{P}[\mathbf{D}_{anomaly}|\pi(P,Q)]$, higher-order model $\pi(P^*,Q^*)$ tends to give even smaller value to the probability of the corresponding higher-order transitions in space $\mathbf{X}^K$. Putting it

altogether, we have:

$$P[\ \mathbf{D}_{normal}\ |\ \pi(P,Q)\ ]$$
$$\leq P[\ \mathbf{D}_{normal}\ |\ \pi(P^*,Q^*)\ ] \tag{65}$$

$$P[\ \mathbf{D}_{anomaly}\ |\ \pi(P,Q)\ ]$$
$$\geq P[\ \mathbf{D}_{anomaly}\ |\ \pi(P^*,Q^*)\ ] \tag{66}$$

2)   The model $\pi[P^*(S),Q^*(S)]$ that takes into account the multi-variate sequence (with return value being the second dimension for example) can do even better the the multi-order model $\pi(P^*,Q^*)$ alone, or:

$$P[\ \mathbf{D}_{anomaly}\ |\ \pi(P^*,Q^*)\ ]$$
$$\geq P\{\ \mathbf{D}_{anomaly}\ |\ \pi[\ P^*(S),Q^*(S)\ ]\ \} \tag{67}$$

6   CONCLUSIONS

In this paper, we proposed a multi-order Markov chain based anomaly detection framework. By monitoring the relative relations between results from the different-order models, we provide a new effective indicator of anomalies. In general, due to the regular and periodical behaviors of cloud server systems, if the probability of test set given the lower-order model exceeds that given the higher-order one, it is implied that unusual events might have occurred in the system and further attentions or actions would be necessary.

Besides, combining multi-dimensional inter-related sequences as a multivariate one into a single model would be another feasible approach to improve the sensitivity of detection. As shown previously, the return value series can be a useful complement to the system call series used the traditional practice.

In addition, with both time and space efficiency of the *Training* and *Testing* algorithm, this approach minimizes the possibility of becoming the source of anomalies itself and is fully capable of online (or real-time) detection. The time consumption of the training stage takes no more than 15 seconds for a training set as large as 1.6 million, and for models up to the third-order combined. To further improve efficiency, various ways such as equivalent space construction by including an artificial state, or binary representation for sparse matrix could significantly mitigate the space complexity problem.

Further efforts may involve the time-inhomogeneous Markov chain, which might be able to build transition matrix separately for any specific time period, as the time-homogeneous assumption stated here could seem too restrictive for time-sensitive systems and anomalies. Model aggregations methods may also be relevant in combining multi-order conditional probabilities into one single value for faster decision-making.

7   ACKNOWLEDGMENT

REFERENCES

[1] Denning, D.E., "An intrusion-detection model," *IEEE Transactions on Software Engineering*. pp. 222 - 232, Feb. 1987
[2] X. D. Hoang, J. Hu, and P. Bertok, "A program-based anomaly intrusion detection scheme using multiple detection engines and fuzzy inference," Journal of Network and Computer Applications, vol. 32, pp. 1219-1228, 2009.
[3] H. T. Elshoush and I. M. Osman, "Alert correlation in collaborative intelligent intrusion detection systemsłA survey," Applied Soft Computing, vol. 11, pp. 4349-4365, 2011.
[4] P. G. Bringas and Y. K. Penya, "Next-Generation Misuse and Anomaly Prevention System Enterprise Information Systems," vol. 19, J. Filipe and J. Cordeiro, Eds., ed: Springer Berlin Heidelberg, pp. 117-129, 2009.
[5] S. Saravanakumar, Amruth Kumar.A, Anandaraj.s, s.Gowtham, "Algorithms Based on Artificial Neural Networks for Intrusion Detection in Heavy Traffic Computer Networks," *Proc. of Int'l Conf. on Advancements in Information Technology*, pp.6-23, 2011
[6] C.V. Raman, Atul Negi, "A Hybrid Method to Intrusion Detection Systems Using HMM," *Distributed Computing and Internet Technology*. pp. 389 - 396, 2005
[7] Yihua Liao, V.Rao Vemuri, "Use of K-nearest Neighbor Classifier for Intrusion Detection," *Computers & Security*. 21(5): 439-448, 2002
[8] Nong Ye, "A Markov Chain Model of Temporal Behavior for Anomaly Detection," *Workshop on Information Assurance and Security*. West Point, NY, June 2000.
[9] Nong Ye, Xiangyang Li, Qiang Chen, Syed Masum Emran, Mingming Xu, "Probabilistic Techniques for Intrusion Detection Based on Computer Audit Data," *IEEE Transactions on Systems, Man, and Cybernetics łPart A: Systems and Humans*. 31(4):266-274, July 2001
[10] Nong Ye, Yebin Zhang, Connie M. Borror, "Robustness of the Markov-Chain Model for Cyber-Attack Detection," *IEEE Transactions on Realiability*. 53(1):116-123, March 2004
[11] Robert Gwadera, Mikhail Atallah, "Markov Models for Identification of Significant Episodes," *Proceedings of the 5th International Conference on Data Mining*. 2005
[12] Wen-Hua Ju, Yehuda Vardi, "A Hybrid High-Order Markov Chain Model for Computer Intrusion Detection," *Journal of Computational and Graphical Statistics*. 10(2): 277-295, 2001
[13] Chuanhuan Yin, Shengfeng Tian, Shaomin Mua, "High-order Markov kernels for intrusion detection," *Neurocomputing*. 71(16-18):3247-3252, 2008
[14] Stephanie Forrest, Steven A, Hofmeyr, Anil Somayaji, Thomas A, Longstaff, "A Sense of Self for Unix Processes," *Proceedings of IEEE Symposium on Security and Privacy*, pp. 120 - 128, 1996
[15] Gaurav Tandon, Philip Chan, "Learning Rules from System Call Arguments and Sequences for Anomaly Detection," *Workshop on Data Mining for Computer Security*. IEEE International Conference on Data Mining, pp. 20 -29, Nov. 19-22, 2003
[16] Wei Wang, Xiaohong Guan, Xiangliang Zhang, Liwei Yang, "Profiling program behavior for anomaly intrusion detection based on the transition and frequency property of computer audit data," *Computers & Security*. 25(7): 539 - 550, 2006
[17] Richard P, Lippmann, David J, Fried, Isaac Graf, et al, "Evaluating Intrusion Detection Systems: The 1998 DARPA Off-line Intrusion Detection Evaluation," *Proceedings of the DARPA Information Survivability Conference and Exposition (DISCEX)*, Vol. 2, pp. 12 - 26, 2000
[18] Markus Stowasser, "Modelling Rain Risk: A Multi-order Markov Chain Model Approach", *The Journal of Risk Finance,* 13(1):45 - 60, 2011

[19] Y. Song, A. D. Keromytis, and S. J. Stolfo, "Spectrogram: A Mixture-of-Markov-Chains Model for Anomaly Detection in Web Traffic," *Proceedings of the Network and Distributed System Security Symposium (NDSS 2009)*, San Diego, CA, USA, February 2009

[20] Wenyao Sha, Yongxin Zhu, Tian Huang, Meikang Qiu, Yan Zhu, Qiannan Zhang, "A Multi-order Markov Chain Based Scheme for Anomaly Detection," *Proceedings of IEEE 37th Annual Computer Software and Applications Conference (COMPSAC) Workshops*, pp. 83-88, 2013

[21] Ja-Min Koo and Sung-Bae Cho, "Interpreting chance for computer security by viterbi algorithm with edit distance," *New Mathematics and Natural Computation*, Vol. 1, No. 3, pp. 421-433, 2005

Min Chen Min Chen (minchen@ieee.org) [M08-SM09] is a professor in School of Computer Science and Technology at Huazhong University of Science and Technology (HUST). He is the Chair of IEEE Computer Society (CS) Special Technical Communities (STC) on Big Data. He was an assistant professor in School of Computer Science and Engineering at Seoul National University (SNU) from Sep. 2009 to Feb. 2012. He was R&D director at Confederal Network Inc. from 2008 to 2009. He worked as a Post-Doctoral Fellow in Department of Electrical and Computer Engineering at University of British Columbia (UBC) for three years. Before joining UBC, he was a Post-Doctoral Fellow at SNU for one and half years. He received Best Paper Award from IEEE ICC 2012, and Best Paper Runner-up Award from QShine 2008. He has more than 180 paper publications, including 85 SCI papers. He serves as editor or associate editor for Information Sciences, Wireless Communications and Mobile Computing, IET Communications, IET Networks, Wiley I. J. of Security and Communication Networks, Journal of Internet Technology, KSII Trans. Internet and Information Systems, International Journal of Sensor Networks. He is managing editor for IJAACS and IJART. He is a Guest Editor for IEEE Network, IEEE Wireless Communications Magazine, etc. He is Co-Chair of IEEE ICC 2012-Communications Theory Symposium, and Co-Chair of IEEE ICC 2013-Wireless Networks Symposium. He is General Co-Chair for IEEE CIT-2012 and Mobimedia 2015. He is General Vice Chair for Tridentcom 2014. He is Keynote Speaker for CyberC 2012 and Mobiquitous 2012. He is a TPC member for IEEE INFOCOM 2013 and INFOCOM 2014. His research focuses on Internet of Things, Big Data, Machine to Machine Communications, Body Area Networks, E-healthcare, Mobile Cloud Computing, Ad Hoc Cloudlet, Cloud-Assisted Mobile Computing, Ubiquitous Network and Services, and Multimedia Transmission over Wireless Network, etc.

Wenyao Sha was born in Shanghai, China in 1989. He received his B.S. degree in Microelectronics from Shanghai Jiao Tong University in 2012. His research interest includes machine learning, data mining, algorithm optimizing, asset pricing and financial forecasting.

Tian Huang Tian Huang (M2013) was born in Shanghai, China in 1986. He received the B.S. degree in Electronics Science and Technology from Shanghai Jiao Tong University in 2008. He is currently pursuing a Ph.D. degree in Electronics Science and Technology at Shanghai Jiao Tong University. His research interest includes Computer Architecture, Embedded System, Multi-media Signal Processing, Parallel Processing, and System on Chip.
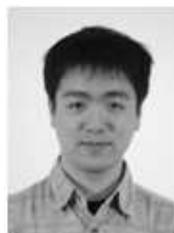
Yongxin Zhu Yongxin Zhu (M 2000, SM2010) is an Associate Professor with the School of Microelectronics (merged into School of Electronic Information and Electrical Engineering recently), Shanghai Jiao Tong University, China. He is a senior member of China Computer Federation and a senior member of IEEE. He received his B.Eng. in EE from Hefei University of Technology, and M. Eng. in CS from Shanghai Jiao Tong University in 1991 and 1994 respectively. He received his Ph.D. in CS from National University of Singapore in 2001. His research interest is in computer architectures, embedded systems, medical electronics and multimedia. He has authored and co-authored over 70 English journal and conference papers and 30 Chinese journal papers. He holds more than 10 approved Chinese patents.