Process Mining Algorithm for Online Intrusion Detection System

Yinzheng Zhong[0000−0001−8477−3956], John Y. Goulermas[0000−0003−0381−124X], and Alexei Lisitsa[0000−0002−3820−643X]

Department of Computer Science, University of Liverpool, UK
sgyzhon5@liverpool.ac.uk, goulerma@liverpool.ac.uk,
a.lisitsa@liverpool.ac.uk

Abstract. In this paper, we consider the applications of process mining in intrusion detection. We propose a novel process mining inspired algorithm to be used to preprocess data in intrusion detection systems (IDS). The algorithm is designed to process the network packet data and it works well in online mode for online intrusion detection. To test our algorithm, we used the CSE-CIC-IDS2018 dataset which contains several common attacks. The packet data was preprocessed with this algorithm and then fed into the detectors. We report on the experiments using the algorithm with different machine learning (ML) models as classifiers to verify that our algorithm works as expected; we tested the performance on anomaly detection methods as well and reported on the existing preprocessing tool CICFlowMeter for the comparison of performance.

Keywords: Intrusion detection · Process mining · Deep learning · Anomaly detection · Cybersecurity.

1 Introduction

With the growth of applications that relies on internet communication, cybercrime became a serious issue that affects many areas. By estimation, about 33 billion records of personal information including addresses, credit card information, or social security numbers etc., will be stolen in 2033 [9]. The intrusion detection systems (IDS) protect computer systems by monitoring the network or system activities. One of the main challenges in the design of IDS is to have fast and robust methods for network traffic assessment to be used for the detection of attacks and malicious behaviour. The *process mining* has risen recently as a promising research direction aiming at systematic developments of the methods for building behavioural or workflow models from event logs [2,11]. The process mining is essentially approaches that takes information (e.g. cases, timestamps and events) from the event logs for building the workflow models (process models) which can then be used for analytical tasks. The process model describes the transitions of events within traces.

While the applications of process mining in the security have been considered, e.g. in [1], its applications in IDS remain largely unexplored. In this paper, we

propose process mining inspired technique to be used at the preprocessing stage to generate a behaviour model, which subsequently be classified as attack/no attack or normal/malicious behaviour by trained machine learning models. There are similar approaches for IDS, for example, based on *data mining* [8] and *machine learning* [3, 6]. In most of the cases, however, these approaches can only detect the threats after features been generated based on flows. In our approach, the process mining is used as the preprocessing step, while machine learning is used as the classifier. Our proposed algorithm for process mining of network data can be seen as a modification of the initial model mentioned in the fuzzy mining algorithm [5]. The latter was modified for better online processing, and techniques such as aggregation and abstraction in fuzzy mining could also be applied. The rest of the paper is organized as follows. In the next section, we give a short outline of the related work. After that, in Section 4 we present the proposed process mining inspired algorithm. The setup for machine learning is discussed in Section 5. Section 6 reports on experiments and Section 7 presents the discussion and outlines the future work.

2   Related Work

The fuzzy mining algorithm was introduced in [5]. The process model is built on the initial model with various filterings and abstractions. The initial model is the high-level description of processes that preserves all relations. We tried to use fuzzy and inductive mining traditionally for intrusion detection by performing conformance checking, but the result is far worse than expected [1]. We get the inspiration from fuzzy mining and modified the algorithm to perform online mining, which can then be used as a preprocessing step for online intrusion detection. The algorithm will be described in Section 4.

CICFlowMeter is a preprocessing tool that generates features, such as bytes per second, inter arrival time, packets per second etc., based on the network flows. The TCP flow terminates at FIN packet, where for UDP connections a timeout value needs to be set. The CICFlowMeter has been introduced in [7]. We compare the performance of CICFlowMeter with our preprocessing algorithm in Section 6.

We use multi-layer perceptron (MLP), long short-term memory (LSTM), convolutional neural network (CNN), and k-nearest neighbours (KNN) in our binary classification and multi-class classification setups. The reason we choose these classifiers is:

- MLP is a simple feedforward model.
- LSTM is a recurrent model that can be applied on time series data.
- CNN works directly on 2D inputs and it has been widely used in image classification problems.
- KNN is an example of traditional distance based classifier.

  For the anomaly detection setup, we used the following outlier detectors.

- Multivariate normal distribution (MND).

- Copula-Based Outlier Detection (COPOD).
- AutoEncoder.
- Angle-Based Outlier Detection (ABOD).
- Clustering-Based Local Outlier Factor (CBLOF).
- Histogram-based Outlier Score (HBOS).
- Isolation Forest (IForest).
- K-nearest Neighbors (KNN).
- Local Outlier Factor (LOF).
- Principal Component Analysis (PCA).

Note that the reason we choose these models as our classifiers is because they have different properties and characteristics, and the purpose of comparing them is just to verify that the preprocessing algorithms works as expected so it can be applied onto different classifiers.

3   Dataset

The dataset we used in this experiment is the CSE-CIC-IDS2018 dataset [10]. The dataset contains common attacks such as Bruteforce, DoS, and Botnet etc. The dataset comes with two formats, the extracted features in CSV spreadsheets and the PCAP binary packet data. We used Tshark to extract necessary attributes (IPs, ports, and flags) of TCP packets from the PCAP data for our algorithm. We also generated dataset with CICFlowMeter using the same PCAP data that were used as the training set for our preprocessing algorithm.

4   Process Mining & Measuring Frequency of Transitions

The packets observed on the wire is the sequence $P = \langle p_i \rangle_{i=1}^n$, where p_i is each individual packet. The observed packets can also form a set of TCP flows $T = \{t_i\}_{i=1}^m$, where each flow t_i can be constructed according to the IP addresses and ports of two hosts (T can also be considered as the event log from a perspective of process mining). Please note that in process mining, a flow would correspond to a *trace*, and both of these terms may be used in this paper interchangeably. We define that a new TCP flow is started when a packet that has flag SYN set but without ACK set (first packet of three-way handshake) is received, also, this initial packet determines the IP addresses and ports of two hosts. For example, the packet has $Source\ IP : Port = IP_1 : PORT_1$ and $Target\ IP : Port = IP_2 : PORT_2$. The bidirectional flows can be reconstructed based on forward direction ($IP_1 : PORT_1 \rightarrow IP_2 : PORT_2$) and backward direction ($IP_2 : PORT_2 \rightarrow IP_1 : PORT_1$). We define the TCP flow as completed when the packet that has the FIN flag or RST flag set is received.

As mentioned above, instead of analysing the flows, we analyse the relations between packets in flows, which is the basic idea of process mining [11]. Before we discuss the algorithm we need to define the concepts of *transitions* and *event classes*.

Given a sequence of packets P, we define a *transition* in P as a pair of consecutive packets (p_i, p_j) within a flow in P.

Here is an example, giving two traces t_1 and t_2, where $t_1 = \langle p_1, p_3, p_5 \rangle$ and flow $t_2 = \langle p_2, p_6 \rangle$, we will get two transitions for t_1: (p_1, p_3) and (p_3, p_5); one transition for t_2: (p_2, p_6). Packets p_1 and p_2 come from two consecutive packets, however, these packets will not be considered as a transition as they belong to different flows.

An *event class* $ec(p)$ of a packet p is the concatenation of enabled flags of a packet followed by an indicator. e.g. 000.SYN.|C, where the last character is an indicator that indicates either the packet is sent from the client or the server. In this case, C indicates the packet is sent from the client.

A *type* of the transition (p_i, p_j) is a pair of corresponding event classes $(ec(p_i), ec(p_j))$. We will also refer to types of transitions as *relations*. e.g. (000-.SYN.|C, 000.ACK.SYN.|S) is a *relation* and which indicates that a packet has SYN flag enabled is followed by a consecutive packet that has ACK and SYN enabled.

We have 23 possible event classes that were observed from the IDS2018 dataset of normal traffic data. We assume these 23 event classes cover the majority of possible flag combinations. All other packets that have flag combinations that were not observed in the dataset can be simply classified as OTHERS as a default rear case handling, or the event classes can be adjusted according to a particular situation.

There are 26 event classes in total, including 23 event classes from the flag data and 3 default classes (START, END and OTHERS). Therefore, there will be $26^2 = 676$ possible *relations* if we assume every classes can be paired with other classes. These 26 event classes are available in Table 1.

Our proposed online algorithm operates as follows. Given a sequence of packets P (even log), the algorithm outputs the sequence (stream) of frequencies of relations observed in the last l packets (for some l), organized in a form of adjacency matrix (26x26). Here, the frequencies of relations observed in the last l packets are process models.

We want to measure the frequency by counting the incoming relations into an adjacency matrix A, however, we will only calculate the frequency based on the last l packets, and the frequency of the transitions needs to be updated per each new packet.

If the initial packet p_1 in trace t_1 carries 000.SYN.|C, then the weight of $A(START, 000.SYN.|C)$ will be increased by one; and if the next packet p_3 in the same trace carries 000.ACK.SYN.|S, then the weight in $A(000.SYN.|C, 000.ACK.SYN.|S)$ will be increased by one.

In our experiments, we have limited the number of packets l that used to calculate the frequency of transitions to 500 by using the sliding window. The limitation here is just our choice based on various experiment and any number is possible to be used here. The sliding window starts from p_1 and covers $\langle p_i \rangle_{i=1}^{l}$, and the frequency of transitions A' is calculated as A/l, then the window will be shifted one step further which covers $\langle p_i \rangle_{i=2}^{l+1}$. This process results in a sequence

Table 1. Possible event classes used.

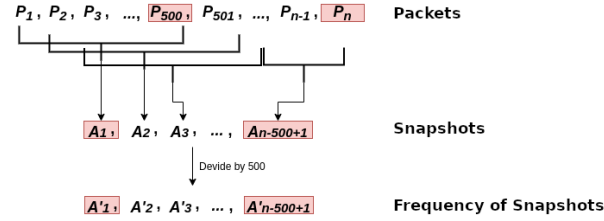| 000.SYN.|C	000.ACK.SYN.|S	000.ACK.|C	000.ACK.PSH.|C
000.ACK.PSH.|S	000.ACK.FIN.|C	000.ACK.|S	000.ACK.FIN.|S
000.ACK.RST.|C	000.ACK.RST.|S	000.RST.|S	000.ACK.PSH.FIN.|S
000.RST.|C	000.CWR.ECE.SYN.|C	000.ECE.ACK.SYN.|S	000.NS.ACK.FIN.|S
000.ACK.PSH.FIN.|C	000.CWR.ACK.PSH.|C	000.CWR.ACK.|C	000.CWR.ACK.|S
000.CWR.ACK.PSH.|S	000.CWR.ACK.RST.|S	000.CWR.ACK.RST.|C	START
END	OTHERS		

$\langle A'_i \rangle_{i=1}^{n-l+1}$ and each A'_i is a snapshot of a process model with l events. In process mining, the events are instances of event classes. The process of producing A is similar to mining the fuzzy model where t_i is traces and P is the event log. However, the modification here is that the last state that is outside the window of each flow t_i was kept in the state table so the START and END tokens will only occur at the beginning and end of a particular TCP flow, not where it begins and end in each sliding window. As the last state is known, the relation can be mined even if the window has already passed the previous event. In other words, the original process mining takes the entire event log P into account, and in that case P generates a single huge adjacency matrix A. However, this is not suitable for online processing, therefore, we keep the states of traces through the entire event log P but limit the process model generation based on l packets only. This keeps all the transition information and makes it suitable for online systems.

For the purpose of performance and ease of use, we used the l-sized buffer to keep the transitions that are inside the sliding window instead of count every transition again in each loop, and we ignored the transition to END, therefore, we only need to update two elements in A for each packet (decrease the frequency for the packet that goes off the buffer and increase the frequency for the packet that goes on the buffer). When computing the frequencies of transitions, we also produced the output for machine learning based on the labelled data provided from the dataset. The dataset provides the source of a certain attack, therefore, if the most recent packet in A_i was sent from or to the IP that was labelled as a certain attack (13 types of attacks in total), A_i will be marked as a model that contains attack. For better demonstration, Fig. 1 of Botnet attacks is provided, where attacks are marked in red colour. The red bar at the bottom of the chart in Fig. 2 indicates the locations of attacks. In summary, we have two outputs from this process, the frequencies of the transitions A'_i and the location of attacks. The pseudocode (Algorithm 1) is given in Appendix A below and Fig. 2 shows the example of the output of 20 relations.
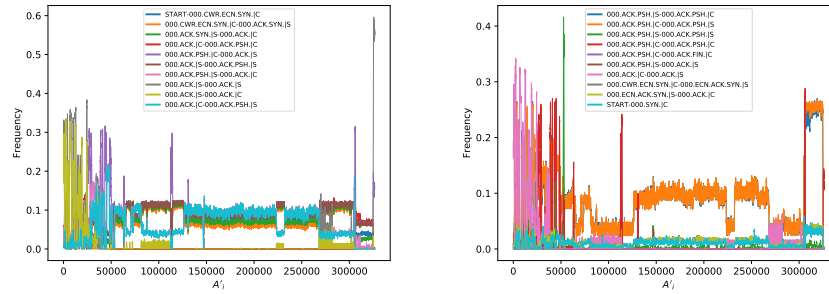
5   Experiments

5.1   Knowledge-based detection

The proposed workflow applicable in knowledge-based (signature-based) IDS is as follows. The IP and flag information of network packets are captured,

Fig. 1. Diagram of algorithm 1. Packets P_{500} and P_n belong to traces that are marked as attacks, therefore, A_1 and $A_{n-500+1}$ are also labelled as attacks for training classifiers.

Fig. 2. The chart show frequency fluctuation under Botnet attacks. 20 out of 676 possible relations are given as the example. 10 relations in the first chart and 10 relations in the second chart. In the first chart, the attack happens between $50,000^{th}$ to $310,000^{th}$ packets (ground truth), and it's clear that the frequency starts to stabilise.

and the flows are reconstructed according to the IPs. The flows go through our preprocessor, then a series of snapshots A_i' matrices are generated. These snapshots will be reshaped if needed and then get fed into the ML classifiers trained to recognize *known* attacks. Finally, the classifier will raise the alarm whenever we got an attack.

Except for the input of CNN, A_i' (26 by 26 matrices) was flattened into 676-dimensional vectors (the elements of the vector are the frequencies of transitions), and which was used as the input for MLP, LSTM, and KNN. For binary classifications, the output data are 2-D vectors where $[1\ 0]^\mathsf{T}$ stands for normal traffic and $[0\ 1]^\mathsf{T}$ stands for intrusions; for multi-class classifications, the output was 14-dimensional one-hot vectors that encode the output to be normal or one of 13 types of attacks. As all of the locations of attacks were marked, the output data can be generated from the algorithm above (Section 4). We have not reduced the dimension as we want the classifiers to handle the input data.

The multi-layer perceptron was constructed with 4 dense layers (676, 128, 128 and 2 units for each layer) followed by a softmax layer; the LSTM was built with 2 layers of LSTM (128 units each), one dense layer with 2 units and one

softmax layer lastly; the CNN model is similar to the model used in [6], however, we increased the input shape to 26-by-26 and added one dense layer (32 units) before the output layer. For multi-class classification, the number of units for the last dense layer had been increased from 2 to 14 for all neural networks. All instances had been tested through 5-fold cross-validation. We have three setups for the LSTM with 50, 100 and 250 timesteps.

The dataset has imbalanced normal and anomalous entries. Therefore, normal data has been added into samples or been removed to match the number of the attack data for binary classifications. This helps to train the neural networks but does not affect the F-score. Also, attack types SQL-Injection and Infiltration have been discarded in binary classifications due to an insufficient amount of data.

5.2   Anomaly-based detection

The proposed workflow for anomaly-based IDS is as follows. The necessary information of network packets are captured, and flows are reconstructed, then the flows go through the preprocessor, which generates snapshots. These steps are identical to the signature-based IDS. The snapshots are reshaped into vectors and fed into outlier detectors, then the outlier detectors provide the outlier scores, where a higher outlier score indicates the data has a higher probability of being an anomaly intrusion.

We use the PyOD python library for anomaly-based detection, and hyperparameters of all outlier detectors remain default. The data format is the same as the one used in the binary classification, where all data have been reshaped to 676-dimensional vectors. For data generated with CICFlowMeter, we did the column-wise normalisation before feeding them into the detectors. We only use the normal data that do not contain any attack to train the detectors, then used mixed data, which contain both normal data and attack data for testing.

6   Results

We compared the results of binary classification in Table 2, and the results of binary classification have also been compared with the results from [6] (Table 4). The LSTM gives worse result compared to other models, especially in multi-class classification (Table 3). Some F1 scores produced by LSTM multi-class classification display NaN (not a number) or 0, meaning these classes have 0 in both precision and recall (i.e. 0 on the diagonal line in the confusion matrix), or 0 in either precision or recall. However, this may indicates that with the preprocessing step, the classifiers do not require the historical data to perform the classification, as the historical data have already been encoded during the process showing in Fig. 1.

We focus on the preprocessing step, so we compare our results with CICFlowMeter in Table 4. Both column uses CNN as the classifier, and the result of our approach is promising.

Table 2. the F1 scores for binary classification.

Attack	MLP	LSTM-50	LSTM-100	LSTM-250	KNN	CNN
FTP-BruteForce	0.9990	0.9976	0.9984	0.9982	**0.9991**	0.9990
SSH-Bruteforce	0.9763	0.9764	0.9763	0.8907	0.9732	**0.9764**
DoS-GoldenEye	0.9212	0.9680	0.7457	0.9498	**0.9821**	0.9434
DoS-Slowloris	0.9945	0.8490	0.8513	0.7770	0.9947	**0.9948**
DoS-SlowHTTP	0.9983	0.9798	0.9937	0.9803	**0.9985**	0.9984
DoS-Hulk	0.7309	0.6732	**0.7731**	0.7313	0.7381	0.7314
DDoS-LOIC-HTTP	**0.9968**	0.7606	0.7129	0.7443	0.8353	0.8406
DDOS-HOIC	**0.9687**	0.7072	0.7922	0.6935	0.7879	0.7559
BruteForce-Web	**0.9962**	0.9588	0.9621	0.9631	0.9789	0.9741
BruteForce-XSS	**0.9985**	0.9676	0.9674	0.9788	0.9868	0.9827
Botnet	0.8168	**0.9644**	0.9137	0.8308	0.8754	0.8623

Table 3. the F1 scores for multi-class classifications.

Attack Type	MLP	LSTM-50	LSTM-100	LSTM-250	KNN	CNN
Normal	0.5533	0.4117	0.5401	0.3115	**0.8306**	0.6457
FTP-BruteForce	0.9734	NaN	NaN	NaN	**0.9991**	0.9743
SSH-Bruteforce	0.9772	0.9254	0.9206	0.9215	0.9753	**0.9774**
DoS-GoldenEye	0.9287	0.9219	0.9209	0.9215	**0.9618**	0.9277
DoS-Slowloris	0.9830	0.2433	0.9573	0.8323	**0.9935**	0.9837
DoS-SlowHTTP	0.9070	0.5056	0.5042	0.5546	**0.9981**	0.8844
DoS-Hulk	0.8241	0.8245	0.8212	0.7611	0.7688	**0.8244**
DDoS-LOIC-HTTP	**0.8755**	0.8754	0.524	0.8668	0.8480	**0.8755**
DDOS-HOIC	0.8188	0.8232	0.2553	0.8078	**0.8253**	0.8203
BruteForce-Web	0.9659	0.0334	NaN	0	0.9679	**0.9687**
BruteForce-XSS	0.9634	NaN	NaN	0	**0.9847**	0.9756
SQL-Injection	0.1765	NaN	NaN	NaN	**0.4941**	0.2908
Infiltration	0.0898	0.0417	0.142	0.0179	**0.4334**	0.0149
Botnet	0.6473	0.6465	0.6295	0.4089	0.6277	**0.6540**

We compared the receiver operating characteristic (ROC) for the anomaly-based intrusion detection setup in Fig. 3. The type of attacks was not separated, so what we have here is the overall performance. It's clear that our algorithm works better in anomaly-based intrusion detection.

We have published the preprocessed data and some of the experiment results online.[1]

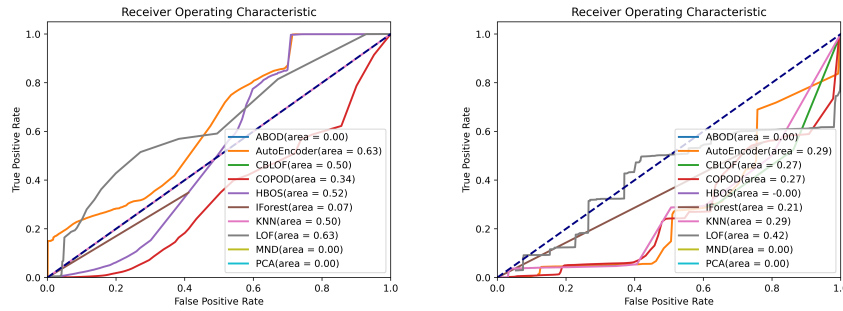# 7   Conclusion & Discussion

We proposed the online process mining algorithm that preprocesses packet data for intrusion detection. The initial process mining algorithm was modified to adapt online process mining, which produces a series of process model snapshots

---

[1] https://zenodo.org/record/5616678

Table 4. comparison between preprocessors.

Attack Type	Our Preprocessor	CICFlowMeter
FTP-BruteForce	**0.9990**	0.98
SSH-Bruteforce	**0.9764**	0.96
DoS-GoldenEye	**0.9434**	0.47
DoS-Slowloris	**0.9948**	0.66
DoS-SlowHTTP	0.9984	**1**
DoS-Hulk	0.7314	**1**
DDoS-LOIC-HTTP	0.8406	**1**
DDOS-HOIC	0.7559	**1**
BruteForce-Web	**0.9741**	0.3
BruteForce-XSS	**0.9827**	0.65
Botnet	0.8623	**1**

Fig. 3. The receiver operating characteristic (ROC) for anomaly-based intrusion detection setup. The first chart shows the performance of our preprocessor, where the second chart shows the performance of CICFlowMeter.

with fixed window sizes. The snapshots were normalized and being used as the input data for machine learning. For signature-based intrusion detection, we used several machine learning models for binary and multi-class classification and yielded high accuracy. Though our preprocessing algorithm does not produce better results for every aspect, the performance was consistently high. On the anomaly-based intrusion detection side, the result is not impressive; however, considering the nature of anomaly-based intrusion detection and the performance of CICFlowMeter, we are satisfied with our preprocessing algorithm. We take it as a starting point to further extend the research on process mining applications in intrusion detection.

Currently, we use snapshots similar to the initial model of the fuzzy miner, and it is possible to apply abstraction on the snapshots easily; therefore, our model can be easily extended with process mining techniques.

As mentioned in Section 6, the classification step does not require any historical snapshots as the historical information has already been packed into the

latest snapshot. This might be more efficient than using a recurrent neural network (RNN) to directly classify the latest n packet data without separate the flows; and it might be more accurate as our approach keeps the state of the flow until connection closed, not just getting information from the last n packets. It needs to be clarified here that this is just a hypothesis.

Because this algorithm is used for preprocessing, other methods for classification can be easily applied. The output of our preprocessor is normalised and can be directly fed into many classifiers or outlier detectors without much modification. Furthermore, the only attribute we used to generate event classes is the flag, so there is still a large set of unconsidered attributes we could use. These could be verified in our future research. Another problem for machine learning on IDS is that when training the neural networks with one dataset but test the accuracy with a different dataset, the accuracy drops massively [4]. We are planning to verify if process-mining based preprocessing can help to resolve such an issue.

A   Pseudocode

Algorithm 1: pseudocode of packet preprocessing.

```
 1  in P = [n];                                              /* load n packets */
 2  in l = 500;                                         /* define the window size */
 3  A = [26 by 26];                               /* a 26 * 26 adjacency matrix */
 4  list_A' = [ ];                                 /* initialise list of A'_i */
 5  list_attacks = [ ];                           /* initialise list of attacks */
    /* a dictionary where the key is the concatenation of IPs and Ports
       ("IP_1 : PORT_1|IP_1 : PORT_1'') of hosts, and the value is the flags of the
       previous packet                                                        */
 6  dict_state_table = {};
 7  buffer = [l];          /* an FIFO buffer that keeps the last l transitions (events) */
    /* initialise with first l packets                                        */
 8  for i = 1 to l do
        /* check if the packet belong to any existing flow                    */
 9      if "IP_1 : PORT_1|IP_2 : PORT_2'' in dict_state_table.key() or
           "IP_2 : PORT_2|IP_1 : PORT_1'' in dict_state_table.key() then
            /* count the transition into A                                     */
10          A[dict_state_table["IP_1 : PORT_1|IP_2 : PORT_2''], current_flags] += 1;
            /* update the state of the flow to the current flags into the dict */
11          dict_state_table["IP_1 : PORT_1|IP_2 : PORT_2''] = current_flags;
12          push current_flags into buffer;
            /* check whether TCP flow terminates                              */
13          if "FIN" in current_flags or "RST" in current_flags then
14              remove key "IP_1 : PORT_1|IP_2 : PORT_2'' from dict_state_table;
15          end
        /* check if new TCP flow starts                                       */
16      else if "SYN" in current_flags and "ACK" not in current_flags then
17          A[dict_state_table["START''], current_flags] += 1;
18          push current_flags into buffer;  /* push the transition (event) into buffer
               */
19      end
20  end
```

```
21  append A/l to list_A';              /* append the frequency of transitions into the list */
22  for i = l + 1 to n do
23  │   if ″IP₁ : PORT₁|IP₂ : PORT₂″ in dict_state_table.key() or
    │      ″IP₂ : PORT₂|IP₁ : PORT₁″ in dict_state_table.key() then
24  │   │   A[pop buffer] −= 1;  /* sub 1 for transition that went outside the window */
25  │   │   A[dict_state_table[″IP₁ : PORT₁|IP₂ : PORT₂″], current_flags] += 1;
26  │   │   dict_state_table[″IP₁ : PORT₁|IP₂ : PORT₂″] = current_flags;
27  │   │   push current_flags into buffer;
    │   │   /* check whether TCP flow terminates                                       */
28  │   │   if ″FIN″ in current_flags or ″RST″ in current_flags then
29  │   │   │   remove key ″IP₁ : PORT₁|IP₂ : PORT₂″ from dict_state_table;
30  │   │   end
    │   /* check if new TCP flow starts                                               */
31  │   else if ″SYN″ in current_flags and ″ACK″ not in current_flags then
32  │   │   A[pop buffer] −= 1;
33  │   │   A[dict_state_table[″START″], current_flags] += 1;
34  │   │   push current_flags into buffer;
35  │   end
36  │   append A/l to list_A';
    │   /* Attack IP is from the labelled data                                        */
37  │   if Attack IP in current_flags then
38  │   │   append i to list_attacks
39  │   end
40  end
    Output: list_A';
    Output: list_attacks;
```

References

1. Van der Aalst, W.M., de Medeiros, A.K.A.: Process mining and security: Detecting anomalous process executions and checking process conformance. Electronic Notes in Theoretical Computer Science **121**, 3–21 (2005)
2. Van der Aalst, W.M., Weijters, A.J.: Process mining: a research agenda (2004)
3. Agarap, A.F.M.: A neural network architecture combining gated recurrent unit (gru) and support vector machine (svm) for intrusion detection in network traffic data. In: Proceedings of the 2018 10th International Conference on Machine Learning and Computing. pp. 26–30. ACM (2018)
4. Al-Riyami, S., Coenen, F., Lisitsa, A.: A re-evaluation of intrusion detection accuracy: Alternative evaluation strategy. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. pp. 2195–2197 (2018)
5. Günther, C.W., Van Der Aalst, W.M.: Fuzzy mining–adaptive process simplification based on multi-perspective metrics. In: International conference on business process management. pp. 328–343. Springer (2007)
6. Kim, J., Shin, Y., Choi, E., et al.: An intrusion detection model based on a convolutional neural network. Journal of Multimedia Information System **6**(4), 165–172 (2019)
7. Lashkari, A.H., Draper-Gil, G., Mamun, M.S.I., Ghorbani, A.A.: Characterization of tor traffic using time based features. In: ICISSp. pp. 253–262 (2017)
8. Lee, W., Stolfo, S.: Data mining approaches for intrusion detection (1998)
9. Norton: 10 cyber security facts and statistics for 2018. `https://us.norton.com/`
10. UNB: Ids 2017 dataset. `https://www.unb.ca/cic/datasets/ids-2017.html`
11. Van Der Aalst, W.: Process mining: discovery, conformance and enhancement of business processes, vol. 2. Springer (2011)