Can process mining help in anomaly-based intrusion detection?

Yinzheng Zhong[0000−0001−8477−3956] and Alexei Lisitsa[0000−0002−3820−643X]

Department of Computer Science, University of Liverpool, UK
y.zhong10@liverpool.ac.uk, a.lisitsa@liverpool.ac.uk

Abstract. In this paper, we consider the naive applications of process mining in network traffic comprehension, traffic anomaly detection, and intrusion detection. We standardise the procedure of transforming packet data into an event log. We mine multiple process models and analyse the process models mined with the inductive miner using ProM [19] and the fuzzy miner using Disco [7]. We compare the two types of process models extracted from event logs of differing sizes. We contrast the process models with the RFC TCP state transition diagram [14] and the diagram [3] by Bishop et al. We analyse the issues and challenges associated with process mining in intrusion detection and explain why naive process mining with network data is ineffective.

Keywords: Anomaly detection · Intrusion detection · Process mining · Cyber security.

1 Introduction

This paper contains some of our earlier works prior to [21]. Our objective is to create an intrusion detection system using process mining (PM) and process models. The models can then be utilised for analytics [2,18]. Network intrusion detection systems (NIDS) are typically based on the following methodologies. First, the features are extracted from the flow-level or connection-level data, and then the features are generated from statistical data, such as the number of packets and the average payload size [9,10]. For classification, these approaches frequently employ data mining, machine learning, and other statistical methods. Second, audit trail methods examine the network's activities, such as the payload content, flag set, or numerical values [15,20]. The issue with the majority of existing techniques is that they are unable to observe the global process structure of a communication.

The paper by Van der Aalst and de Medeiros [1] proposed to use process mining on audit trails and conformance checking for anomaly detection; however, additional research for NIDS is required. Our experiment is predicated on the premise that the process models can be mined from event logs using PM techniques, and then new cases will be validated using conformance checking. The anomaly will then be identified based on the conformance score. We complied with the directive by converting network packet data to event logs, which may facilitate PM adaptation.

We employ ProM for inductive mining algorithm, which is related to [1]. The inductive miner will mine models in the form of Petri nets. The extracted models will then be utilised for additional analysis, such as conformance checking. In contrast, we will compare fuzzy mining as an alternative PM algorithm [8].

The paper is structured as follows. First, we will look at related works that use similar approaches; second, we will demonstrate the dataset we use and how the packet data are converted into the event log; then, we will discuss the experimental setups and compare the models. After that, we show the result of conformance checking and discuss the reason for the inefficiency of PM for IDS. Finally, we introduce a novel design of a preprocessing technique that can be used in NIDS which is inspired by PM.

2   Related Works

The technique introduced in [13] is used on industrial control systems (ICS) that identifies anomalous behaviours and cyber-attacks using ICS data logs and the conformance checking analysis technique. The paper demonstrated the preprocessing of ICS data logs for PM and the result of anomaly detection. The inductive mining algorithm discovers their process model, and as a result, the average F-score for detecting attacks across two datasets is 0.81. A case is categorised as an anomaly if it does not produce a fitness score of 1. The paper [13] is not focused on NIDS.

On top of [1], papers [11, 12] provide expanded discussions on PM for IDS, discussing potential benefits of using PM for IDS, comparing different mining algorithms (such as the heuristic miner and the α-algorithm), and displaying cyber-attacks detected in various industrial sectors, etc. Nonetheless, the precise method for implementing PM for IDS remains undetermined.

The paper [4] uses Inductive Miner Infrequent algorithm to detect anomalous behaviors on the e-commerce platform. The data is processed based on the log recorded from the ModSecurity, which is a open source web application firewall, for the purpose of adapting PM. The events are mainly consist of URLs and activated ModSecurity rules. The provided average fitness scores show that anomalies have lower scores, however, they do not have a clear measurement for F-score or accuracy.

The methodology described in [5] is based on PM and employs social network analysis metrics to identify anomalous behaviour. Instead of performing conformance testing, this method analyses social network metrics to identify anomalies. The log of normal traces is extracted from the process model using the workflow Petri net designer (WoPeD) process modelling tool [6]; the role-trace access control matrix, which is based on the normal traces and the role-activity access control matrix, is then created; and finally, the social network is constructed using the role-trace access control matrix and all logs. The experiment's F-score is greater than 0.92, per the outcome.

In general, PM applications in intrusion detection are not well established.

3   Data Prepossessing

Table 1. The event log segment. The three zeros in flags are the reserved bits.

Case_ID	Timestamp	Src_IP	Dst_IP	Scr_port	Dst_port	Flags	Side
151043	2017/07/04 14:00:35.190179000	192.168.10.5	68.67.178.110	52892	80	000.SYN.	C
151043	2017/07/04 14:00:35.222535000	68.67.178.110	192.168.10.5	80	52892	000.ACK.SYN.	S
151043	2017/07/04 14:00:35.222586000	192.168.10.5	68.67.178.110	52892	80	000.ACK.	C
151043	2017/07/04 14:00:35.237412000	192.168.10.5	68.67.178.110	52892	80	000.ACK.PSH.	C
151043	2017/07/04 14:00:35.270301000	68.67.178.110	192.168.10.5	80	52892	000.ACK.	S
151043	2017/07/04 14:00:35.270305000	68.67.178.110	192.168.10.5	80	52892	000.ACK.	S
151043	2017/07/04 14:00:35.467062000	68.67.178.110	192.168.10.5	80	52892	000.ACK.	S
151043	2017/07/04 14:00:35.467285000	68.67.178.110	192.168.10.5	80	52892	000.ACK.PSH.	S
151043	2017/07/04 14:00:35.467347000	192.168.10.5	68.67.178.110	52892	80	000.ACK.	C
151043	2017/07/04 14:00:35.467352000	68.67.178.110	192.168.10.5	80	52892	000.ACK.PSH.	S
151043	2017/07/04 14:00:35.511515000	192.168.10.5	68.67.178.110	52892	80	000.ACK.	C
151043	2017/07/04 14:00:45.461285000	192.168.10.5	68.67.178.110	52892	80	000.ACK.	C
151043	2017/07/04 14:00:45.466144000	68.67.178.110	192.168.10.5	80	52892	000.ACK.FIN.	S
151043	2017/07/04 14:00:45.466173000	192.168.10.5	68.67.178.110	52892	80	000.ACK.	C
151043	2017/07/04 14:00:45.466198000	192.168.10.5	68.67.178.110	52892	80	000.ACK.FIN.	C
151043	2017/07/04 14:00:45.493763000	68.67.178.110	192.168.10.5	80	52892	000.ACK.	S
151043	2017/07/04 14:00:45.498542000	68.67.178.110	192.168.10.5	80	52892	000.ACK.	S
281008	2017/07/05 17:03:45.498235000	192.168.10.16	23.194.141.47	51226	443	000.SYN.	C
281008	2017/07/05 17:03:45.521284000	23.194.141.47	192.168.10.16	443	51226	000.ACK.SYN.	S
281008	2017/07/05 17:03:45.521360000	192.168.10.16	23.194.141.47	51226	443	000.ACK.	C
281008	2017/07/05 17:03:45.521561000	192.168.10.16	23.194.141.47	51226	443	000.ACK.PSH.	C
281008	2017/07/05 17:03:45.544708000	23.194.141.47	192.168.10.16	443	51226	000.ACK.	S
281008	2017/07/05 17:03:45.545025000	23.194.141.47	192.168.10.16	443	51226	000.ACK.PSH.	S
281008	2017/07/05 17:03:45.545073000	192.168.10.16	23.194.141.47	51226	443	000.ACK.	C
281008	2017/07/05 17:03:45.561805000	192.168.10.16	23.194.141.47	51226	443	000.ACK.PSH.	C
281008	2017/07/05 17:03:45.624241000	23.194.141.47	192.168.10.16	443	51226	000.ACK.	S
281008	2017/07/05 17:03:45.820846000	192.168.10.16	23.194.141.47	51226	443	000.ACK.PSH.	C
281008	2017/07/05 17:03:45.843822000	23.194.141.47	192.168.10.16	443	51226	000.ACK.	S
281008	2017/07/05 17:03:45.921777000	23.194.141.47	192.168.10.16	443	51226	000.ACK.PSH.	S
281008	2017/07/05 17:03:45.964684000	192.168.10.16	23.194.141.47	51226	443	000.ACK.	C
281008	2017/07/05 17:03:46.802581000	192.168.10.16	23.194.141.47	51226	443	000.ACK.PSH.	C
281008	2017/07/05 17:03:46.825827000	23.194.141.47	192.168.10.16	443	51226	000.ACK.	S
281008	2017/07/05 17:03:46.827025000	23.194.141.47	192.168.10.16	443	51226	000.ACK.PSH.	S
281008	2017/07/05 17:03:46.827041000	192.168.10.16	23.194.141.47	51226	443	000.ACK.	C
281008	2017/07/05 17:03:53.457141000	192.168.10.16	23.194.141.47	51226	443	000.ACK.FIN.	C
281008	2017/07/05 17:03:53.480155000	23.194.141.47	192.168.10.16	443	51226	000.ACK.PSH.	S
281008	2017/07/05 17:03:53.480156000	23.194.141.47	192.168.10.16	443	51226	000.ACK.FIN.	S
281008	2017/07/05 17:03:53.480264000	192.168.10.16	23.194.141.47	51226	443	000.RST.	C
281008	2017/07/05 17:03:53.480268000	192.168.10.16	23.194.141.47	51226	443	000.RST.	C

In our experiment, we utilised the IDS2017 dataset [17]. The dataset contains common attacks including Bruteforce, DoS, and Botnet, among others. The dataset includes PCAP packet data as well as CSV data sheets with analytical features that have been preprocessed. Focusing on packet-level detection, we will only utilise binary packet data and extract necessary data from this dataset using TShark. The processed and transformed data must then be applied to PM as event logs. In this paper, we only consider TCP data. We will keep two attributes for the event logs, the first attribute is the packet flags information;

the second will be the are labels which indicate whether a packet is sent from the server (S) or the client (C).

First, we filtered out invalid packets that do not use the TCP protocol or were missing data in part. Second, we reconstructed each case based on the socket pair (source IP_Port and destination IP_Port) and assigned a unique case ID to each flow. A *case* in PM is essentially an instance of a series of actions/activities. In network traffic, we define it as a series of packets within a connection; therefore, in subsequent contexts, we may refer to the case as the network flow. The TCP header flags will then be converted into human-readable strings, such as ACK (for ACK flag sets) or ACK.FIN (for ACK and FIN flags set). We define a complete flow as one in which the SYN flag is set at the beginning and the RST or FIN flag is set at the end; thus, all cases that do not begin with SYN or do not contain RST/FIN will be discarded.

The IDS2017 dataset includes numerous PCAP binary files, each of which stores packets from a specific host. Some files only contain normal traffic data, so we combine the processed event logs from these files to create a comprehensive normal event log (the complete-log). Flows involved in attacks are also transformed separately into anomalous event logs. All normal and anomalous event logs are saved as CSV data sheets. These files will constitute our final datasets, which can be fed directly into the PM tools.

Table 1 shows an example of a segment of the event log. Note that IPs and Ports are not used in PM as they are used for determine the case ID only. The data columns used in the experiment are case IDs, timestamps, header flags and side labels. We select header flags and side labels as our attributes as they are non-numerical and the transitional information can be constructed with these data, which is suitable for PM. The transition here is defined as (a, b) where a and b are consecutive events within a case.

4 Experiment Setups

We retained every hyperparameter as default in both ProM and Disco in PM. The first reason is that understanding the problems is more important than looking through different mining hyperparameters. Second, we want to mine authentic models from the event logs we provide because rare cases are not necessarily anomalies; also we know these rare cases are normal as they come from the normal event log.

For the purpose of comparison and comprehension, we sub-sampled the complete-log into different smaller-sized datasets and mined the process model from these subsets with both process discovery algorithms. Here are the setups for inductive miner in ProM.

1. Process model with 5 flows #1.
2. Process model with 5 flows #2.
3. Process model with 100 flows #1.
4. Process model with 100 flows #2.

5. Process model with 20k flows #1.
6. Process model with 20k flows #2.
7. Process model with 100k flows #1.
8. Process model with 100k flows #2.
9. Process model of complete set.

Similar setups are used for fuzzy miner with Disco; however, Disco is limited to 50k cases. Subsets with fewer than 50k cases are identical to inductive miner subsets. The details are presented below.

1. Process model with 5 flows #1.
2. Process model with 5 flows #2.
3. Process model with 100 flows #1.
4. Process model with 100 flows #2.
5. Process model with 20k flows #1.
6. Process model with 20k flows #2.
7. Process model with 50k flows #1.
8. Process model with 50k flows #2.

5   Model Comparisons

5.1   Process Models

Now, let us compare process models mined using inductive miner for setups 1), 3) and 5) in Fig. 1, Fig. 2 and Fig. 3 respectively. These setups represent the small, medium and large event logs. Event logs that are larger than 20k cases do not show significant difference in our experiment.

Fig. 1. The Petri net mined with 5 cases.

The model mined with 5 cases is a small-log instance for testing how the number of cases impacts the generality and accuracy of the model. Nine different event classes are observed from these 5 cases, and the model mined from 20k cases contains 19 observed event classes. An *event class* is essentially the name of an observed event; in our process model, it is the concatenated strings of both attributes from a case, for example, 000.SYN|C.

Fig. 2. The Petri nets mined with 100 cases.

Fig. 3. The Petri nets mined with 20k cases.

From these two models, we see better accuracy on the model mined with 5 cases and better generality on the model mined with 20k cases. The model mined with 20k cases discovers the trace which terminates after the first packet of the three-way handshake (000.SYN|C); however, the same model also permits impossible transitions, such as entering the data-transmission stage without a three-way handshake. The model mined with 100 cases has a good trade-off between the models mined with 5 cases and 20k cases. We compare the process models mined with fuzzy miner with the same event logs from setup 1), 3) and 5) in Fig. 4, Fig. 5 and Fig. 6 respectively.

In our case, the fuzzy models provide a better balance between precision and generality for network traffic data. We evaluate the generality of small models and the precision of large models. The model mined with 5 cases is still quite accurate, similar to the inductive model; however, the differences between the models mined with 20,000 cases are greater. For instance, the model mined with 20,000 cases of Disco does not suffer from the handshake skipping issue. The fuzzy mining algorithm theoretically does not generate an edge unless the transition is observed in the event log. In other words, the large fuzzy model achieves greater generality, but retains a good accuracy.

5.2 Diagrams

Now we compare all process models with two state diagrams, the RFC TCP state transition diagram provided in [14] and the improved diagram produces by Bishop et al. in [3]. We use this comparison for a better understanding of how

Fig. 4. The fuzzy models mined with 5 cases.

TCP works, what PM can observe, and how the results of process mining are comparable with the common descriptions of TCP protocol by state diagrams.

The RFC TCP state transition diagram describes a TCP flow in general terms. The diagram depicted the establishing, established, and closing stages of a TCP flow. This diagram depicts the establishing stage as a three-way handshake. The diagram lacks information regarding the established stage. The closing stage consists of two states on each side of communication (sending out FIN and waiting for ACK). This diagram has a much higher abstraction than any of the models mined, and all we can compare are the handshake and closing phases. In the diagram, cases such as disconnecting during a handshake are not permitted.

The information shown by the state diagram of Bishop et al. is closer to the real traffic data as the analysis is based on real network traffic observed on the system calls. Transitions with RST set are coloured orange; transitions that have SYN set are coloured green; transitions that have FIN set are coloured blue; others are coloured black. We have flags in the diagram such as 'Arsf', where 'A' stands for ACK set, and 'a' stands for ACK clear; 'R' stands for RST set, and 'r' stands for RST clear;'S' stands for SYN set, and 's' stands for SYN clear; 'F' stands for FIN set, and 'f' stands for FIN clear. We refer readers to the original report in [3] for more details.

This state diagram has significantly more detail than the original RFC diagram. It displays additional traces and information about event classes containing the RST flag. START is not directly connected to PSH or END, similar to

Fig. 5. The fuzzy models mined with 100 cases.

Fig. 6. The fuzzy models mined with 20k cases.

the process models discovered using Disco. Event names in our process models contain the labels S and C, which indicate whether a packet was sent upstream or downstream; however, the state diagram preserves the states of the RFC TCP state diagram, and it does not reflect where a packet was sent from.

Based on our understanding of the the diagram, we make the transition matrix in Table 2 that describe possible transitions and possible traces. For example, row 1 column 2 shows that after a packet with SYN set, we can possibly get a packet with ACK set next.

```
                           +---------+ --------\        active OPEN
                           | CLOSED  |          \       ----------
                           +---------+<---------\   \   create TCB
                              |     ^            \   \  snd SYN
                 passive OPEN |     |   CLOSE      \   \
                 ----------   |     |   ---------   \   \
                 create TCB   |     |   delete TCB   \   \
                           V  |                       \   \
                        +---------+          CLOSE     |    \
                        | LISTEN  |          ---------  |     |
                        +---------+          delete TCB |     |
              rcv SYN      |     |    SEND              |     |
              ----------   |     |    ------            |     V
+---------+  snd SYN,ACK  /      \   snd SYN        +---------+
|         |<------------------                ------------------->|         |
|  SYN    |             rcv SYN                                   |  SYN    |
|  RCVD   |<-----------------------------------------------|  SENT  |
|         |                 snd ACK                              |         |
|         |------------------                                    |         |
+---------+  rcv ACK of SYN  \                /  rcv SYN,ACK      +---------+
|           --------------    |              |   -----------
|                  x          |              |     snd ACK
|                             V              V
|  CLOSE                    +---------+
|  -------                  |  ESTAB  |
|  snd FIN                  +---------+
|              CLOSE          |     |    rcv FIN
V            -------          |     |    -------
+---------+     snd FIN      /       \   snd ACK        +---------+
|  FIN    |<------------------                ------------------->|  CLOSE  |
| WAIT-1  |------------------                                     |  WAIT   |
+---------+      rcv FIN       \                                  +---------+
| rcv ACK of FIN   -------     |                                    CLOSE  |
| --------------   snd ACK     |                                    ------- |
V         x                    V                                   snd FIN V
+---------+                  +---------+                          +---------+
|FINWAIT-2|                  | CLOSING |                          | LAST-ACK|
+---------+                  +---------+                          +---------+
|           rcv ACK of FIN |                      rcv ACK of FIN |
| rcv FIN     -------------- |     Timeout=2MSL    -------------- |
| -------              x     V     -----------         x         V
\ snd ACK                  +---------+delete TCB      +---------+
------------------------->|TIME WAIT|---------------->| CLOSED  |
                          +---------+                 +---------+
```
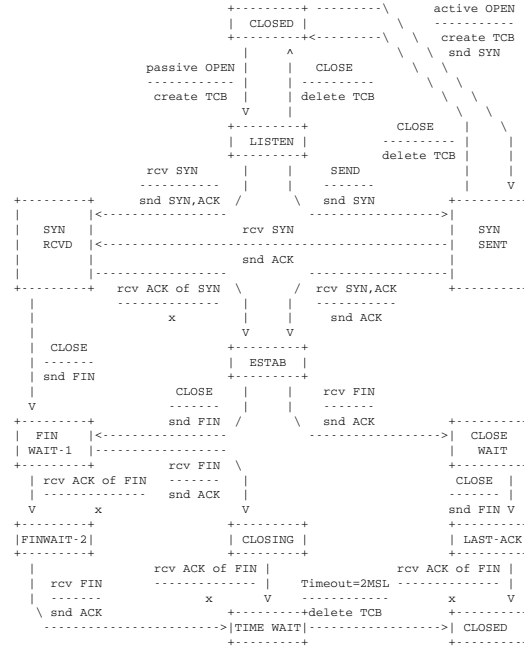
Fig. 7. RFC TCP state transition diagram [14].

Table 2. The adjacency matrix of state diagram by Bishop et al.

	SYN.	ACK.	ACK.SYN.	ACK.RST.	FIN.	DATA	ACK.FIN.	RST.	START	END
SYN.	1	1	1	1				1		1
ACK.							1	1		
ACK.SYN.	1	1	1	1	1					
ACK.RST.										1
FIN.	1	1			1		1	1		
DATA	1	1		1	1		1	1		
ACK.FIN.	1	1			1					
RST.									1	1
START	1							1	1	
END										

5.3   Overall Comparison

In Table 3, we compare the level of generality of various discussed models. A general model should permit normal, unobserved behaviours to pass compliance testing. Compared are only rare normal flows that exist in actual network traffic data. The examples illustrate some infrequent flows. The table columns display models, such as an inductive model mined with five traces, a fuzzy model mined with one hundred traces, etc. The row illustrates rare cases that we anticipate being observed in the models. If the case is observed in any process model or
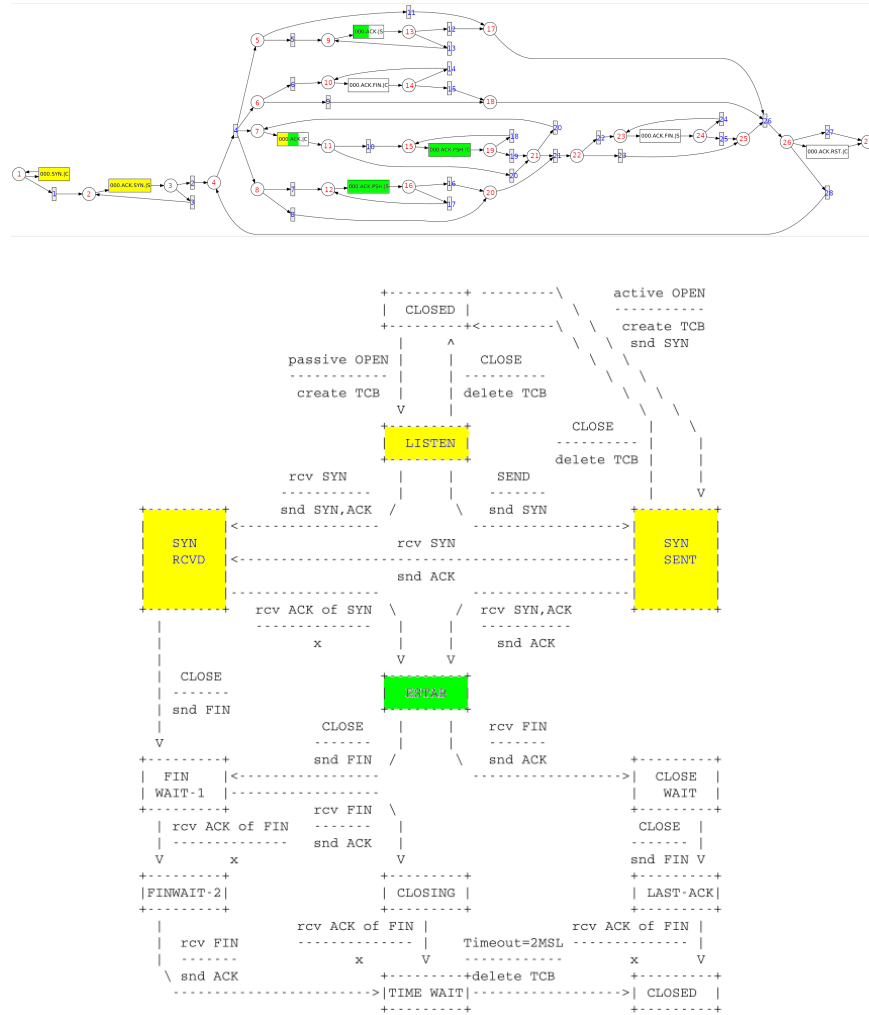
Fig. 8. State diagram by Bishop et al. [3].

diagram, the cell is marked Y(es); otherwise, the cell is marked N(o). Following the inductive model in terms of generality is the fuzzy model. This result is anticipated due to the algorithmic design, in which an inductive miner explores patterns such as concurrencies, yielding a large number of possible traces that are not present in the event log. Alternatively, RFC diagrams are general descriptions of TCP states, and both diagrams are limited to predefined stages such as LISTEN, SYN-SENT, and ESTABLISHED, among others.

Table 3. Generality Comparison.

	Ind. 5t	Fuzzy 5t	Ind. 100t	Fuzzy 100t	Ind. 20kt	Fuzzy 20k	RFC Diag.	Bishop
PSH in established stage.	Y	Y	Y	Y	Y	Y	N	N
Duplicate pakages.	Y	Y	Y	Y	Y	Y	N	Y
Reset during handskake.	N	N	Y	Y	Y	Y	N	Y
Send of receive SYN after estanblished.	N	N	N	N	Y	Y	N	N
Reset during closing	Y	N	Y	N	Y	Y	N	Y

Table 4. Accuracy Comparison.

	Ind. 5t	Fuzzy 5t	Ind. 100t	Fuzzy 100t	Ind. 20kt	Fuzzy 20k	RFC Diag.	Bishop
Data transmission without handshake	N	N	Y	N	Y	N	N	N
RST or FIN before handshake	N	N	Y	N	Y	N	N	N
Loop through Close and Established stages	N	N	Y	Y	Y	Y	N	N
Sending same packets infinitely without response from the other side	Y	Y	Y	Y	Y	Y	N	N

Fig. 9. Stage comparison between process model and diagram.

We contrast accuracy in Table 4. An accurate model that depicts normal behaviours should not allow abnormal traces to pass the compliance checking. We assume a trace is abnormal based on typical use cases because the behaviour of a flow can vary depending on the implementation of an application. In contrast to Table 3, we DO NOT anticipate any of the cases in Table 4 to fit any process model or diagram. We achieve greater accuracy with fuzzy models, and the diagrams are the most accurate. Cases involving infinite loops are a common issue with process models. Because diagrams adhere to predefined stages, looping through the close stage and the established stage cannot occur. Due to the fact

that ACK typically occurs after receiving any packet, a link will be created between this packet and the ACK packet, causing loops between stages. To better illustrate this limitation of process models, we colour the process model in Fig. 1 into three stages; thus, we can directly compare it with the diagrams at the level of stages. Fig. 9 depicts the coloured process model. The yellow nodes correspond to events from the three-way handshake stage; the green nodes correspond to events from the data transmission stage; and the uncoloured nodes correspond to events from the termination stage.

The event class "000.ACK|C" occurs in all three stages of this example, whereas "000.ACK|S" occurs in the established and closing stages. This is the reason for the loops between stages being created, and the fuzzy model has the same problem. We label each packet with C or S to ensure that the information regarding which side of the terminals sent a particular packet will not be lost in PM, and by adding this information, we have also prevented the creation of additional loops. Although this information is not directly present in the packet data, it should be added to event logs by processing the raw packet data. Therefore, our experiment can also be improved by including information about the stages.

On the basis of this observation, the question can be posed: what exactly can PM extract from event logs? In our case, without prior knowledge of the TCP protocol and preprocessing the packet data prior to adding them to the event log, the information of C/S and stages will be lost, resulting in a greater number of loops in mined modesl and a greater degree of difficulty for the analysis.

6   Conformance Checking

Table 5. Fitness scores of conformance checking.

Normal	BruteForce	DoS	Heartbleed	CoolDisk	Dropbox	PortscanNmap	Web	Botnet_ARES	Port_Scan_DDos
1	1	1	1	1	1	1	1	1	1

Using the inductive models which is mined with a 20k event log (the same event log from setup 5), we performed conformance checking and calculate the fitness score with ProM. A score of 1 indicates that all cases perfectly match the process model; otherwise, the score will be less than 1. The method for performing conformance testing is described in detail in [16]. The average fitness scores for multiple setups are displayed in Table 5. Using a 20k-case event log, we mine the normal process model; then, we perform conformance checking with various categories including another normal log and all attacks. The normal column represents the fitness score of an additional subset distinct from the one used to generate the model. On Zenodo, all preprocessed data will be accessible[1].

---

[1] https://doi.org/10.5281/zenodo.6646875

As all cases exhibit perfect model fit, it is impossible to identify the anomaly. Observable traces are a limitation of related works that mine the normal model from only a few dozens or hundreds of cases, resulting in a relatively accurate model. However, network flows can be significantly more complex; therefore, mining a process model from deviated flows will result in a significantly more general model. As long as the flows are being routed on the network, regardless of whether they are attacks or normal traffic, they always comply with TCP, and attacks typically do not exploit the protocol itself. We believe that the frequency distribution of transitions is more crucial for anomaly detection in IDS. An anomaly may not be viewed as a particular transitional change, but rather as the global frequency structure.

7 Conclusion

We believe that using PM and conformance checking directly for network intrusion detection is ineffective. We compare models, and based on our observations, we know that it is difficult to detect anomalies due to the generality of the process model. In addition, we explained why using PM for network traffic anomaly detection is ineffective. A benefit of PM is that it encodes the global process structure, allowing parallelism to be discovered. This may be important for detecting distributed denial of service (DDoS), Botnet, and brute force 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. Bishop, S., Fairbairn, M., Norrish, M., Sewell, P., Smith, M., Wansbrough, K.: Tcp, udp, and sockets: rigorous and experimentally-validated behavioural specification: Volume 1: Overview. Tech. rep., University of Cambridge, Computer Laboratory (2005)
4. Bruno, M., Ibañez, P., Techera, T., Calegari, D., Betarte, G.: Exploring the application of process mining techniques to improve web application security. In: 2021 XLVII Latin American Computing Conference (CLEI). pp. 1–10 (2021)
5. Ebrahim, M., Golpayegani, S.A.H.: Anomaly detection in business processes logs using social network analysis. Journal of Computer Virology and Hacking Techniques **18**(2), 127–139 (2022)
6. Eckleder, A., Freytag, T.: Woped a tool for teaching, analyzing and visualizing workflow nets. Petri Net Newsletter **75**, 3–8 (2008)
7. Günther, C.W., Rozinat, A.: Disco: Discover your processes. In: BPM (2012)
8. 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)
9. Hsu, Y.F., He, Z., Tarutani, Y., Matsuoka, M.: Toward an online network intrusion detection system based on ensemble learning. In: 2019 IEEE 12th international conference on cloud computing (CLOUD). pp. 174–178. IEEE (2019)

10. Lee, W., Stolfo, S.: Data mining approaches for intrusion detection (1998)
11. Mishra, V.P., Dsouza, J., Elizabeth, L.: Analysis and comparison of process mining algorithms with application of process mining in intrusion detection system. In: 2018 7th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions)(ICRITO). pp. 613–617. IEEE (2018)
12. Mishra, V.P., Shukla, B.: Process mining in intrusion detection-the need of current digital world. In: International Conference on Advanced Informatics for Computing Research. pp. 238–246. Springer (2017)
13. Myers, D., Suriadi, S., Radke, K., Foo, E.: Anomaly detection for industrial control systems using process mining. Computers & Security **78**, 103–125 (2018)
14. Postel, J.: Transmission control protocol (1981)
15. Roesch, M., et al.: Snort: Lightweight intrusion detection for networks. In: Lisa. vol. 99, pp. 229–238 (1999)
16. Rozinat, A., Van der Aalst, W.M.: Conformance checking of processes based on monitoring real behavior. Information Systems **33**(1), 64–95 (2008)
17. UNB: Ids 2017 dataset. `https://www.unb.ca/cic/datasets/ids-2017.html`
18. Van Der Aalst, W.: Process mining: discovery, conformance and enhancement of business processes, vol. 2. Springer (2011)
19. Verbeek, H., Buijs, J., Van Dongen, B., van der Aalst, W.M.: Prom 6: The process mining toolkit. Proc. of BPM Demonstration Track **615**, 34–39 (2010)
20. Wespi, A., Dacier, M., Debar, H.: Intrusion detection using variable-length audit trail patterns. In: International Workshop on Recent Advances in Intrusion Detection. pp. 110–129. Springer (2000)
21. Zhong, Y., Goulermas, J.Y., Lisitsa, A.: Process mining algorithm for online intrusion detection system. arXiv preprint arXiv:2205.12064, Also to appear in proceeding: 2021 International Conference on Software Testing, Machine Learning and Complex Process Analysis (2022)