Online Transition-Based Feature Generation for Anomaly Detection in Concurrent Data Streams

Yinzheng Zhong[a] and Alexei Lisitsa[b]

Department of Computer Science, University of Liverpool, Liverpool, U.K.

Keywords: Intrusion Detection Systems, Process Mining, Machine Learning.

Abstract: In this paper, we introduce the transition-based feature generator (TFGen) technique, which reads general activity data with attributes and generates step-by-step generated data. The activity data may consist of network activity from packets, system calls from processes or classified activity from surveillance cameras. TFGen processes data online and will generate data with encoded historical data for each incoming activity with high computational efficiency. The input activities may concurrently originate from distinct traces or channels. The technique aims to address issues such as domain-independent applicability, the ability to discover global process structures, the encoding of time-series data, and online processing capability.

1 INTRODUCTION

Anomaly detection in data analysis typically refers to the discovery of uncommon observations of patterns that differ considerably from the majority of the data and do not adhere to a well-defined concept of normal behaviour. Chandola et al. (Chandola et al., 2009) introduce anomaly detection applications in many areas, such as intrusion detection, fraud detection and industrial damage detection. This paper will introduce a generic technique for extracting features from activity changes (transitions) for use in machine learning and signal processing.

Our paper is based on the paper from Zhong et al. (Zhong et al., 2022), which uses a process mining related technique for network intrusion detection. The technique in (Zhong et al., 2022) is inspired by process mining (van Der Aalst et al., 2003; Van Der Aalst, 2011) algorithms that discover process models from event logs. There are procedures involved in every aspect of our daily lives, from the operations of large businesses to the management of private households. In the industrial sector, one can find both the production of automobiles and the fulfilment of customer orders. The procedure or series of activities for achieving a goal is known as the process. We use a network-based intrusion detection system as an illustration of our technique in the context where a network flow of multiple packets is treated as a process.

(Zhong et al., 2022) introduced the feature generation algorithm and the result for intrusion detection but did not introduce other capabilities of the algorithm. Our paper extends the technique of (Zhong et al., 2022) and explore deeper into the technique itself. This paper adds the generality that enables standardised input from applications in different domains, on top of already existing yet introduced capabilities of discovering global process structure that may aid in anomaly detection in concurrent processes, the packet-level (event-level) processing for online detection, and time-series information encoding with reasonable computational complexity.

An intrusion detection system (IDS) is utilised to detect and classify security policy violations and attacks. Depending on the purpose of the system, we have network-based intrusion detection systems (NIDS) and host-based intrusion detection systems (HIDS). The NIDS is usually deployed on infrastructures like routers and switches to detect intrusions by monitoring network activities. The HIDS, on the other hand, inspects each individual system for any unauthorised file modifications, abnormal network activities, or suspicious behaviours.

In the following section, we will first explore some related works that focus on intrusion detection. As (Zhong et al., 2022) is closely related to the intrusion detection domain, we will understand the problem better and discuss what benefits the algorithm from

[a] https://orcid.org/0000-0001-8477-3956

[b] https://orcid.org/0000-0002-3820-643X

Zhong, Y. and Lisitsa, A.
Online Transition-Based Feature Generation for Anomaly Detection in Concurrent Data Streams.
DOI: 10.5220/0011726700003405
In *Proceedings of the 9th International Conference on Information Systems Security and Privacy (ICISSP 2023)*, pages 576-582
ISBN: 978-989-758-624-8; ISSN: 2184-4356

(Zhong et al., 2022) provides. Then we summarise the problems in Section 3 that TFGen is able to solve. The technical details of TFGen will be presented in section 4, and finally, we will discuss the possible applications of TFGen and some known issues of this technique.

2 RELATED WORK

From the detection method perspective, signature-based intrusion detection systems (SIDS) and anomaly-based intrusion detection systems (AIDS) are typically the two types of intrusion detection systems. The SIDS uses patterns to detect intrusions, or machine learning algorithms are trained with labelled data and used to classify whether an intrusion has occurred. Snort (Roesch et al., 1999) is an example of a SIDS that detects intrusions using predefined rules. Also, data mining (Lee and Stolfo, 1998; Borkar et al., 2019; Ashraf et al., 2018), machine learning (Agarap, 2018; Hsu et al., 2019; Roshan et al., 2018; Mirsky et al., 2018), and other statistical methods (Vijayasarathy et al., 2011; Lee and Xiang, 2000; David and Thomas, 2015) exist. The machine learning models are trained with binary or multi-class data in the case of SIDS. For the example of AIDS, Zavrak and Iskefiyeli uses Autoencoder and Support Vector Machines (SVM) for anomaly detection (Zavrak and İskefiyeli, 2020); Abdelmoumin et al. use ensemble learning (Abdelmoumin et al., 2022) for anomaly detection in Internet of Things (IoT). The machine learning models are trained with normal data only (one-class training) in the case of AIDS.

In general, SIDS checks for incoming data characteristics that are comparable to known threats, whereas AIDS analyses the deviation between incoming data and normal behaviour, and the outcome is determined based on whether the outlier score is above the threshold. Typically, the SIDS accuracy measurement is F-score, while the AIDS accuracy measurement is the receiver operating characteristic (ROC) and area under the curve (AUC). The ROC curve is a graph that displays how well a classification model performs across all classification thresholds. The advantage of AIDS is that it is capable of detecting zero-day attacks; however, the disadvantage of AIDS is that it normally has a higher false positive rate (FPR) than SIDS.

We must note that although some publications are proposed to be about AIDS, they do not precisely adhere to the notion of AIDS. For example, (Althubiti et al., 2018) use LSTM for intrusion detection, and (Anton et al., 2019) use machine learning for intrusion detection in industrial network. These techniques produce unreasonably high accuracy and very low FRP and claim to be anomaly-based, but they are signature-based. The survey (Khraisat et al., 2019) shows the concept of AIDS accurately; however, the majority of the papers to which the survey refers are not related to AIDS. We can see the same problem in other surveys (Maseer et al., 2021).

From the detection speed perspective, there are two types of intrusion detection methods, online and offline. Online detection monitors network activity in real time in order to detect threats as quickly as possible. Offline detection typically examines the data logged and is executed manually by the administrator or at a predetermined interval. When discussing online intrusion detection, we anticipate the response time between an attack and the activation of an alarm to be as short as possible, which is why we consider packet-level detection methods in our approach. Numerous techniques employ packet-level detection. However, some techniques, primarily those based on data mining or machine learning, do not detect packet-level intrusions. The first reason is that the commonly used popular datasets lack packet-level information, like the well-known KDD'99 and the improved NSL-KDD datasets (Tavallaee et al., 2009). These datasets provide statistical values at the flow level, i.e., the data is generated only after a socket is closed or timed out. For instance, the Destination Bytes feature indicates the total number of bytes transferred from the source to the destination in a single socket; obviously, this feature cannot be extracted before the socket closes or terminates. For interested readers, (Dhanabal and Shantharajah, 2015) provides details on all features included with the KDD dataset. Second, the efficiency of these systems is insufficient to support packet-level detection.

There are examples of packet-level detection systems. A good example of packet-level detection is presented in (Mirsky et al., 2018). Damped incremental statistics are utilised to generate packet-level feature vectors in real time. By retaining the prior statistical value, the value can be incrementally updated with the most recent packet data. In addition to updating the previous values, the decay function devalues older data. Statistical information between RX and TX is also extracted by implementing the 2-D statistics. There also exist pattern-based packet-level IDSs that examine audit trails for network data, including (Roesch et al., 1999; Wespi et al., 2000). In general, these IDSs look for specific activities or a sequence of activities and then makes a determination based on the defined patterns. Apart from NIDS, the Variable-Length Audit Trail Pattern approach proposed by Wespi et al. (Wespi et al., 2000) is used for HIDS. It captures system commands such as file-open and file-close, then maps them into sequences of characters based on the translation table. A sequence of characters will then be divided into variable-length subsequences using the Teiresias algorithm (Rigoutsos and Floratos, 1998). The subsequences are compared with the training sequences for calculating the boundary coverage. Finally, the algorithm will determine the intrusion with the longest group of uncovered events.

3 EXISTING PROBLEMS

We summarise three main issues with some of the techniques used for intrusion detection. The first is the lack of ability to perform online detection; the second is that some packet-level detection techniques have difficulty applying to encrypted data; third, most techniques lack the ability to detect the global process structures.

Usually, packet-level systems are incapable of decrypting and analysing encrypted payloads. Examples include the case-based agent (Schwartz et al., 2002), which uses case-based reasoning on packet XML data; the previously introduced Snort, which uses predefined rules to check for intrusions, and the technique (Wang et al., 2020), which converts bytes of packets to grayscale images and then uses hierarchical network structure for classification.

Some techniques use recurrent neural networks for intrusion detection, such as (Hwang et al., 2019), which uses LSTM to classify a time-series of raw packets. On small network devices, resources for training and running LSTM are not always available. Using flow-level data is another option, but this level of detection is not what we seek.

Packet-level IDS with the use of historical information addresses the issue with encrypted data; however, we are now facing another issue that may result in poor performance for attacks such as DoS/DDoS and brute force attacks. These attacks are possible with a high number of concurrent connections. Theoretically, each connection may appear completely normal; therefore, the attack cannot be identified if we focus on the information provided by a single connection. These concurrent connections may be identical to the previously identified data, and a signature-based IDS may have good performance; however, any modification to the network flow will render these IDSs ineffective.

The motivation is to use a new technique to address the issues mentioned above. Process mining is intended for business process model discovery and analysis, which is capable of encoding the global process structure. The activities are recorded in the event log that can be used for process mining in the future. The obvious problem is that the collection of these activities could take days or even weeks, and the event log is then used to determine the process model. This may be appropriate for offline intrusion detection, but it cannot be used traditionally for online anomaly detection. However, we believe the ability to observe global process structure is essential for detecting attacks such as botnet, DoS/DDoS, and brute force.

Online conformance checking is available in the later research (van Zelst et al., 2019), but anomaly detection is limited to conformance checking in the process mining domain. Zhong and Lisitsa have done tests in (Zhong and Lisitsa, 2022) as a naive approach to use process mining on network data and then tried to detect anomalies with conformance checking. The results have turned out not promising. An interested reader may find further experimental results and explanations in (Zhong and Lisitsa, 2022).

4 ONLINE FEATURE GENERATION ON CONCURRENT DATA STREAMS

Here we add a more detailed, accurate and general version of the algorithm applicable to various types of streams based on Zhong et al.'s algorithm (Zhong et al., 2022).

Generally, we attempt to utilise the global flow discovery capability of process mining but design it online. As mentioned previously, process mining analyses the relationships between packets in flows and encodes the global flow structure into the process model rather than analysing the flows themselves. The resulting algorithm will be a transition-based preprocessor that takes streams of activities as the input and produces a series of adjacency/transition matrices. These matrices have historical information encoded.

Before discussing the algorithm, it is necessary to re-define the concepts of transitions and event classes, whose traditional definitions in process mining may vary slightly.

Given a sequence of events P, we define a *transition* in P as a pair of consecutive events (p_i, p_j) within the same case. The transition is referred to as the precedence relation, and P can be treated as the event log.

A *trace* is a series of event names. A *case* is a trace

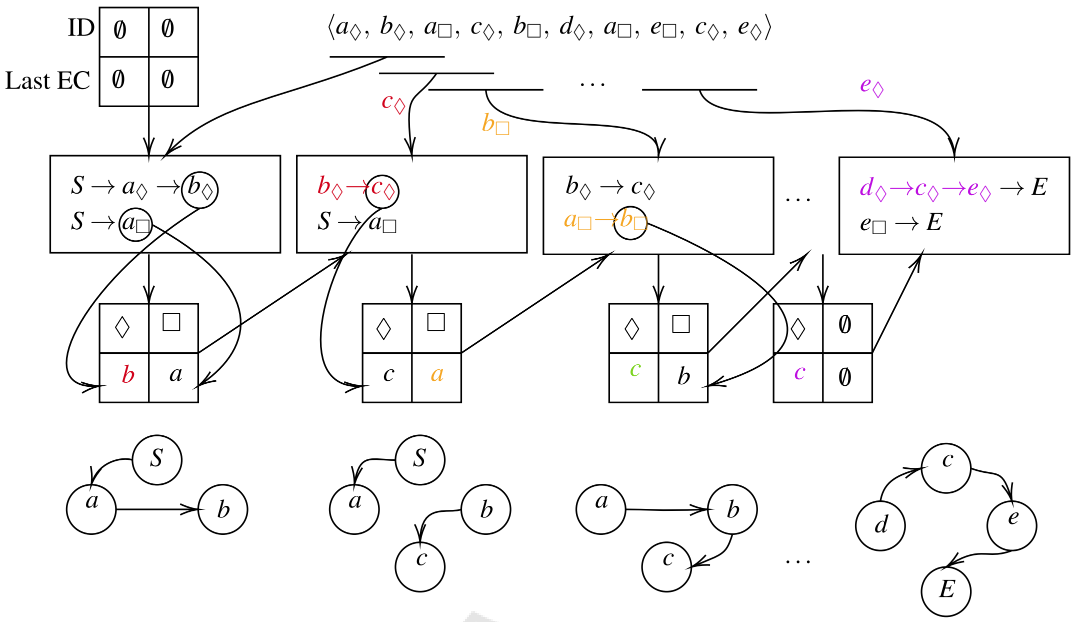
ID
Last EC
∅ ∅
∅ ∅
$\langle a_\Diamond, b_\Diamond, a_\Box, c_\Diamond, b_\Box, d_\Diamond, a_\Box, e_\Box, c_\Diamond, e_\Diamond \rangle$
$c_\Diamond$
$b_\Box$
$e_\Diamond$
$S \to a_\Diamond \to b_\Diamond$
$S \to a_\Box$
$b_\Diamond \to c_\Diamond$
$S \to a_\Box$
$b_\Diamond \to c_\Diamond$
$a_\Box \to b_\Box$
$d_\Diamond \to c_\Diamond \to e_\Diamond \to E$
$e_\Box \to E$
◊ □
b a
◊ □
c a
◊ □
c b
◊ ∅
c ∅
S a b
S a b c
a b c
c d e E
...

Figure 1: Generation of features using TET. S represents the SOT (start of the trace) token, while E represents the EOT (end of the trace) token. The frequent occurrence of an event class within the window causes the weights of nodes and edges to increase; however, the weights are not reflected in this diagram.

instance that has been executed. A trace, for example, is a predefined procedure/process for producing a certain type of medication. This type of medication can be manufactured multiple times, resulting in numerous cases. The event log is comprised of the logs from the production of various types of medications. The case is also referred to as a flow in this paper.

Here is an example, giving a series of events $P = \langle p_1, p_2, p_3, p_4, p_5 \rangle$ with two flows t_1 and t_2, where flow $t_1 = \langle p_1, p_3, p_5 \rangle$ and flow $t_2 = \langle p_2, p_4 \rangle$, 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_4). p_1 and p_2 are two consecutive events; however, these events will not be considered as a transition as they belong to different flows.

An *event class* $ec(p)$ is the name of an event p, and normally it is the concatenated string of non-numerical attribute data. In the example of network traffic, it is the enabled flags of packets and whether the packet comes from the server of the client (Zhong et al., 2022). An event is the executed instance of an event class.

The temporal event table (TET), formerly known as the state table in (Zhong et al., 2022), is a data structure used to prevent the loss of information on transitions for historical data not covered by the sliding window. We call it TET in this paper to eliminate the confusion. Let us consider a smaller scaled example which has an event log of 10 events that are involved in 2 concurrent traces where trace $\Diamond = \langle a_\Diamond, b_\Diamond, c_\Diamond, d_\Diamond, c_\Diamond, e_\Diamond \rangle$ and trace $\Box = \langle a_\Box, b_\Box, a_\Box, e_\Box \rangle$; we have 5 event classes $\{a, b, c, d, e\}$ and assuming our sliding window size l is 3.

The TET has a header, which is typically the case ID, and the event class is stored beneath the case ID. In Figure 1, the TET is initialised with the initial window, and the Last EC value is updated based on the most recent packet. For each incoming packet, the system uses the case ID from the incoming packet as a key to retrieve the previous event. For instance, $c_\Diamond$ has case $\Diamond$, which exists in TET, and the event class record for $\Diamond$ in TET is b; the system creates the relation (b, c) and then updates the record in TET to the last observation c. When the traces reach their end position, the records will be null, and the associated key/ID in the TET will be removed. TET works in conjunction with the sliding window, and the resulting output is transition matrices with size n^2 where n is the number of observed event classes.

TET, in combination with a sliding window buffer that retains the previous l transitions, eliminates the need to generate a graph for each window and the need to search for the previous event with the same trace ID as the incoming event. TET stores the event history and maintains an $O(1)$ computational complexity for processing each activity. TET can be expanded to support variable-length historical events logging, and sliding windows are compatible with process mining techniques such as trace clustering and abstraction (Günther and Van Der Aalst, 2007;

Song et al., 2008).

The main difference between TFGen and traditional process mining is that process mining focus on process model generation and analytical method like conformance checking on process models, whereas TFGen is designed for online processing and feature generation for machine learning. Comparing TFGen to other feather generators for IDS, not only TFGen has the potential to address issues that have been mentioned, but it also generates features based on non-numerical data and attributes instead of numerical data. The TFGen implementation as a Python package can be found on Github[1]. This implementation is capable of processing around 80,000 events per second using a single thread of an Intel Core i5-12600K processor[2].

5 DISCUSSION

This novel feature generator was originally developed for NIDS, and the report on its performance can be found in (Zhong et al., 2022). Some results presented in (Zhong et al., 2022) show AUC under 0.5, and we believe this is due to the fact that some transition frequencies stabilise under attack. Therefore, the attack data have lower variance and may be characterised as attacks by some outlier detectors.

Further improvements obviously can be made. Given the generality and flexibility of the algorithm, the position for our paper is that TFGen may be applicable to the domain of transition-based problems or data that can be mapped to discrete space. Here are some instances.

- It is applicable to HIDS based on system calls or kernel operations (Liu et al., 2018; Byrnes et al., 2020; Kadar et al., 2019). The calls are provided as events, and each process will generate a distinct case. With a larger-scaled environment, agents can be deployed on multiple systems for concurrent data collection from a cluster of hosts; TFGen is ideally applicable as long as cases can be modelled and the lengths of cases are finite.
- Computer vision and sensor-based security systems (Ding et al., 2018; Luo et al., 2018) that detect and monitor a series of activities for health and safety measurement. Instead of using the current approaches, the series of classified activities can be modelled as cases where each case can be produced by specific personnel. Each case consists of events that have several attributes, such as gesture, department and gender. The benefit could be better overall performance, and the behaviours of multiple personnel are encoded.
- Anomaly detection in the operation of critical infrastructure (Gauthama Raman et al., 2019), where TFGen can be used in conjunction with numerical sensor readings to encode time-series activity data.

These applications are based on the hypothesis that TFGen supports all standard event logs as long as processes can be converted to an event log, and the performance and practicability of using TFGen in these areas can be open research questions for future work.

To demonstrate this, we conduct a quick experiment to evaluate the performance of HIDS using the dataset of API calls captured by Cuckoo Sandbox (Nunes, 2018; Nunes et al., 2019). Since the dataset does not include a native event log, event logs are generated based on the timestamps and API names. Events are extracted from logs of all processes, then combined and sorted based on the timestamps.

To reduce the dimensionality, we only generate transition matrices based on a limited number of the most frequent event classes out of over 260 observable event classes. We call the event classes that fall within the limited range of visible (frequent) event classes, and we call other event classes hidden (infrequent) event classes. Hidden event classes are counted into the default event class ”Other” for frequency calculation. the setups are available below.

- t100_ipca0: Limiting the visible event classes to 100.
- t50_ipca0: Limiting the visible event classes to 50.
- t25_ipca0: Limiting the visible event classes to 25.
- t10_ipca0: Limiting the visible event classes to 10.

Figure 2 is the result of utilising the Convolutional Autoencoder for unsupervised learning on data generated by TFGen. The outlier factor of a case is determined by the event with the highest outlier factor. The Convolutional Autoencoder is able to process over 28,000 events/s during inference using batch size 32 on a single RTX 2070S graphics card. Details of the constructed event logs and the implementation can be accessed on Zenodo[3]. The link also provides a document that shows extensive details of this experiment, including other experimental setups and performance benchmarks.

Further analysis and experimentation will be conducted, and a better dataset containing native event log data may be utilised. In contrast to a native event

[1] https://github.com/yinzheng-zhong/TFGen

[2] Using the provided NIDS dataset on Github

[3] https://doi.org/10.5281/zenodo.7154396

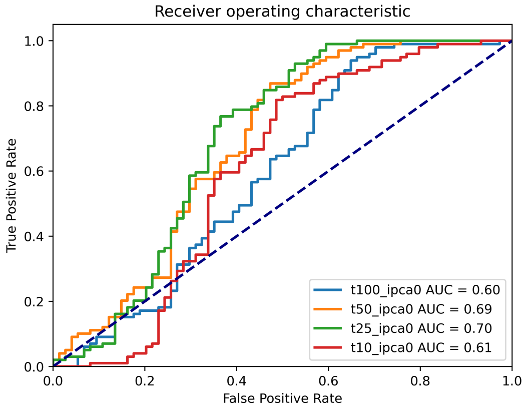

Figure 2: Performance using Convolutional Autoencoder.

log, this generated log may not provide a true representation of concurrent processes.

These are disadvantages of TFGen, for instance, a smaller window size may result in sparse matrices, necessitating an adjustment of the window size based on the problem and anomaly detector. When event classes contain numerous attributes, such as the HIDS dataset used in (Zhong et al., 2022), the output matrices may be of high dimension. Using dimension reduction techniques such as incremental principal component analysis (IPCA) or a limited number of event classes is possible.

To clarify, the purpose of this research is not to demonstrate the high accuracy and low FPR of any experiment but rather to demonstrate that the generalised approach has the potential to function in other domains. Due to the fact that a high FPR is associated with the poor practicality of existing AIDS in general, this algorithm may provide an alternative strategy.

REFERENCES

Abdelmoumin, G., Rawat, D. B., and Rahman, A. (2022). On the performance of machine learning models for anomaly-based intelligent intrusion detection systems for the internet of things. *IEEE Internet of Things Journal*, 9(6):4280–4290.

Agarap, A. F. M. (2018). 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*, pages 26–30. ACM.

Althubiti, S. A., Jones, E. M., and Roy, K. (2018). Lstm for anomaly-based network intrusion detection. In *2018 28th International Telecommunication Networks and Applications Conference (ITNAC)*, pages 1–3.

Anton, S. D. D., Sinha, S., and Dieter Schotten, H. (2019). Anomaly-based intrusion detection in industrial data with svm and random forests. In *2019 International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, pages 1–6.

Ashraf, N., Ahmad, W., and Ashraf, R. (2018). A comparative study of data mining algorithms for high detection rate in intrusion detection system. *Annals of Emerging Technologies in Computing (AETiC), Print ISSN*, pages 2516–0281.

Borkar, G. M., Patil, L. H., Dalgade, D., and Hutke, A. (2019). A novel clustering approach and adaptive svm classifier for intrusion detection in wsn: A data mining concept. *Sustainable Computing: Informatics and Systems*, 23:120–135.

Byrnes, J., Hoang, T., Mehta, N. N., and Cheng, Y. (2020). A modern implementation of system call sequence based host-based intrusion detection systems. In *2020 Second IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA)*, pages 218–225. IEEE.

Chandola, V., Banerjee, A., and Kumar, V. (2009). Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):1–58.

David, J. and Thomas, C. (2015). Ddos attack detection using fast entropy approach on flow-based network traffic. *Procedia Computer Science*, 50:30–36.

Dhanabal, L. and Shantharajah, S. (2015). A study on nsl-kdd dataset for intrusion detection system based on classification algorithms. *International journal of advanced research in computer and communication engineering*, 4(6):446–452.

Ding, L., Fang, W., Luo, H., Love, P. E., Zhong, B., and Ouyang, X. (2018). A deep hybrid learning model to detect unsafe behavior: Integrating convolution neural networks and long short-term memory. *Automation in Construction*, 86:118–124.

Gauthama Raman, M., Somu, N., and Mathur, A. P. (2019). Anomaly detection in critical infrastructure using probabilistic neural network. In *International Conference on Applications and Techniques in Information Security*, pages 129–141. Springer.

Günther, C. W. and Van Der Aalst, W. M. (2007). Fuzzy mining–adaptive process simplification based on multi-perspective metrics. In *International conference on business process management*, pages 328–343. Springer.

Hsu, Y.-F., He, Z., Tarutani, Y., and Matsuoka, M. (2019). Toward an online network intrusion detection system based on ensemble learning. In *2019 IEEE 12th international conference on cloud computing (CLOUD)*, pages 174–178. IEEE.

Hwang, R.-H., Peng, M.-C., Nguyen, V.-L., and Chang, Y.-L. (2019). An lstm-based deep learning approach for classifying malicious traffic at the packet level. *Applied Sciences*, 9(16):3414.

Kadar, M., Tverdyshev, S., and Fohler, G. (2019). System calls instrumentation for intrusion detection in embedded mixed-criticality systems. In *4th International Workshop on Security and Dependability of Critical Embedded Real-Time Systems (CERTS 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

Khraisat, A., Gondal, I., Vamplew, P., and Kamruzzaman, J. (2019). Survey of intrusion detection systems: techniques, datasets and challenges. *Cybersecurity*, 2(1):1–22.

Lee, W. and Stolfo, S. (1998). Data mining approaches for intrusion detection.

Lee, W. and Xiang, D. (2000). Information-theoretic measures for anomaly detection. In *Proceedings 2001 IEEE Symposium on Security and Privacy. S&P 2001*, pages 130–143. IEEE.

Liu, M., Xue, Z., Xu, X., Zhong, C., and Chen, J. (2018). Host-based intrusion detection system with system calls: Review and future trends. *ACM Computing Surveys (CSUR)*, 51(5):1–36.

Luo, Z., Hsieh, J.-T., Balachandar, N., Yeung, S., Pusiol, G., Luxenberg, J., Li, G., Li, L.-J., Downing, N. L., Milstein, A., et al. (2018). Computer vision-based descriptive analytics of seniors' daily activities for long-term health monitoring. *Machine Learning for Healthcare (MLHC)*, 2:1.

Maseer, Z. K., Yusof, R., Bahaman, N., Mostafa, S. A., and Foozy, C. F. M. (2021). Benchmarking of machine learning for anomaly based intrusion detection systems in the cicids2017 dataset. *IEEE access*, 9:22351–22370.

Mirsky, Y., Doitshman, T., Elovici, Y., and Shabtai, A. (2018). Kitsune: an ensemble of autoencoders for online network intrusion detection. *arXiv preprint arXiv:1802.09089*.

Nunes, M. (2018). Dynamic Malware Analysis kernel and user-level calls.

Nunes, M., Burnap, P., Rana, O., Reinecke, P., and Lloyd, K. (2019). Getting to the root of the problem: A detailed comparison of kernel and user level data for dynamic malware analysis. *Journal of Information Security and Applications*, 48:102365.

Rigoutsos, I. and Floratos, A. (1998). Combinatorial pattern discovery in biological sequences: The teiresias algorithm. *Bioinformatics (Oxford, England)*, 14(1):55–67.

Roesch, M. et al. (1999). Snort: Lightweight intrusion detection for networks. In *Lisa*, volume 99, pages 229–238.

Roshan, S., Miche, Y., Akusok, A., and Lendasse, A. (2018). Adaptive and online network intrusion detection system using clustering and extreme learning machines. *Journal of the Franklin Institute*, 355(4):1752–1779.

Schwartz, D., Stoecklin, S., and Yilmaz, E. (2002). Case-based agents for packet-level intrusion detection in ad hoc networks. In *Proc, of the 17th Int. Symp. on Computer and Information Sciences*, volume 7, page 59.

Song, M., Günther, C. W., and Van der Aalst, W. M. (2008). Trace clustering in process mining. In *International conference on business process management*, pages 109–120. Springer.

Tavallaee, M., Bagheri, E., Lu, W., and Ghorbani, A. A. (2009). A detailed analysis of the kdd cup 99 data set. In *2009 IEEE symposium on computational intelligence for security and defense applications*, pages 1–6. Ieee.

Van Der Aalst, W. (2011). *Process mining: discovery, conformance and enhancement of business processes*, volume 2. Springer.

van Der Aalst, W. M., Ter Hofstede, A. H., Kiepuszewski, B., and Barros, A. P. (2003). Workflow patterns. *Distributed and parallel databases*, 14(1):5–51.

van Zelst, S. J., Bolt, A., Hassani, M., van Dongen, B. F., and van der Aalst, W. M. (2019). Online conformance checking: relating event streams to process models using prefix-alignments. *International Journal of Data Science and Analytics*, 8(3):269–284.

Vijayasarathy, R., Raghavan, S. V., and Ravindran, B. (2011). A system approach to network modeling for ddos detection using a naive bayesian classifier. In *2011 Third International Conference on Communication Systems and Networks (COMSNETS 2011)*, pages 1–10. IEEE.

Wang, B., Su, Y., Zhang, M., and Nie, J. (2020). A deep hierarchical network for packet-level malicious traffic detection. *IEEE Access*, 8:201728–201740.

Wespi, A., Dacier, M., and Debar, H. (2000). Intrusion detection using variable-length audit trail patterns. In *International Workshop on Recent Advances in Intrusion Detection*, pages 110–129. Springer.

Zavrak, S. and İskefiyeli, M. (2020). Anomaly-based intrusion detection from network flow features using variational autoencoder. *IEEE Access*, 8:108346–108358.

Zhong, Y., Goulermas, J. Y., and Lisitsa, A. (2022). Process mining algorithm for online intrusion detection system. *arXiv preprint arXiv:2205.12064*.

Zhong, Y. and Lisitsa, A. (2022). Can process mining help in anomaly-based intrusion detection? *arXiv preprint arXiv:2206.10379*.