Log2NS: Enhancing Deep Learning Based Analysis of Logs With Formal to Prevent Survivorship Bias

Charanraj Thimmisetty   Praveen Tiwari   Didac Gil de la Iglesia
Nandini Ramanan   Marjorie Sayer   Viswesh Ananthakrishnan
Claudionor Nunes Coelho Jr*

Advanced Applied AI Research
Palo Alto Networks
https://www.paloaltonetworks.com/

Abstract

Analysis of large observational data sets generated by a reactive system is a common challenge in debugging system failures and determining their root cause. One of the major problems is that these observational data suffers from survivorship bias. Examples include analyzing traffic logs from networks, and simulation logs from circuit design. In such applications, users want to detect non spurious correlations from observational data, and obtain actionable insights about them. In this paper, we introduce log to Neuro-symbolic (Log2NS), a framework that combines probabilistic analysis from machine learning (ML) techniques on observational data with certainties derived from symbolic reasoning on an underlying formal model. We apply the proposed framework to network traffic debugging by employing the following steps. To detect patterns in network logs, we first generate global embedding vector representations of entities such as IP addresses, ports, and applications. Next, we represent large log flow entries as clusters that make it easier for the user to visualize and detect interesting scenarios that will be further analyzed. To generalize these patterns, Log2NS provides an ability to query from static logs and correlation engines for positive instances, as well as formal reasoning for negative and unseen instances. By combining the strengths of deep learning and symbolic methods, Log2NS provides a very powerful reasoning and debugging tool for log-based data. Empirical evaluations on a real internal data set demonstrate the capabilities of Log2NS.

---

*Corresponding author: ccoelho@paloaltonetworks.com

1 Introduction

Survivorship bias Wald (1980) refers to systematic error about our understanding of the world, where we analyze data only based on success cases, omitting consideration (on purpose or not) of the failing cases. One of the most known cases is pictured in Fig. 1 [2] when Abraham Wald attempted to reduce bomber losses due to enemy fire in World War II. During the analysis, he suggested that the bullet marks (hypothetically represented as red dots) showed only cases where the airplanes could land safely, whereas portions without red dots were due to cases when the bombers crashed. Abraham Wald recommended reinforcing portions without red dots, contrary to the original belief that the areas with red dots would need to be reinforced as they showed areas where enemy fire hit the bombers.

There are three takeaways from this example that we will address in this paper.

- It is very easy to analyze complex scenarios through visualizations of observational data based on positive (red dots) and negative (lack of red dots) scenarios;
- Observational data will consist mostly of positive examples;
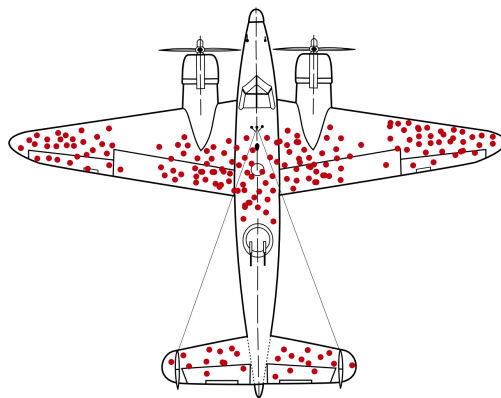- If we cannot get insights from failures or negative scenarios, our analysis may be compromised.

Figure 1: Suvivorship bias

By observational data, we are referring in this paper to data collected over time from a reactive system Harel & Pnueli (1985) - a system that responds to changing actions and conditions, influenced by logical rules and structures. The collected data consists of sequences of inputs and outputs of the system, or sometimes just the outputs, which we denote by *logs* of the system. We assume an associated formal mathematical model, sometimes referred to simply as the *model*. We assume here the formal model encompasses all of the rules and constraints of the reactive system.

We consider that both the real reactive system and the model are capable of generating log data. However, the formal model by construction is a conceptual representation of the real world, which may contain further restrictions on how the reactive system behaves over time. For example, in a real world we may have limitations on what inputs a reactive system may see in the future based on previous outputs, such as in the case of intelligent agents Sutton & Barto (2011).

In numerous Machine Learning and Deep Learning based studies of log data Dangut *et al.* (2020); Fontaine *et al.* (2019); Fang *et al.* (2020); Calabrese *et al.* (2020), it is common for a user who wants to better understand the underlying behavior of the system, or automatically analyze input-output responses of the system, to obtain actionable insights through the collected logs. However, by construction, these logs give rise to survivorship bias Shermer (2014), simply because they reflect observable behaviors of the reactive system over a limited time and limited scenarios, with only

---

[2]credits to By Martin Grandjean (vector), McGeddon (picture), Cameron Moll (concept) - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=102017718

normal cases or with at most a very limited number of rare cases Mehta (2020); Takakis *et al.* (2019). Some of the representative examples where validation and analysis based on observational logs can lead to catastrophic consequences are the Pentium bug Pratt (1995), and the crash of flight AF-447 BOTTYÁN & PALIK (2010).

When analyzing logs, several techniques have been used in the past to reduce the effect of this bias of observable data, such as the use of constrained random simulation Huang *et al.* (2021) or generation of synthetic training data Nikolenko (2019). However, it eventually becomes too hard if not impossible to collect data on certain rare scenarios, and even attempting to creating synthetic data may become a complex task Haixiang *et al.* (2017).

Logs can consists of vast amounts of data, and it is usually useful to determine if one has seen anomalies in the logs, usually occurring as a post processing step Omar *et al.* (2013) or during data collection Mehta (2020). Although machine learning models are useful to summarize and find anomalies in the data in a probabilistic way, because data is biased during data generation, a user may be interested to determine if a scenario is ever possible to occur. Although this question may not be suitable for a machine learning or deep learning model, it can be answered by a symbolic proof system based on a formal model of the reactive system Clarke *et al.* (1996); Kropf (2013); Gupta (1992); Bernardi *et al.* (2021).

This paper presents *Log2NS (Log-to-neural-symbolic)*, a framework that enables users to reason about logs from a reactive systems by using machine learning and formal techniques. This solution can be best represented in Fig. 2 as a revisit of the figure originally presented at Hoehndorf *et al.* (2017), and it can be seen as an instance of using neural-symbolic systems Garcez *et al.* (2008); Lamb *et al.* (2020). The framework first analyzes logs by computing embedding vectors on the
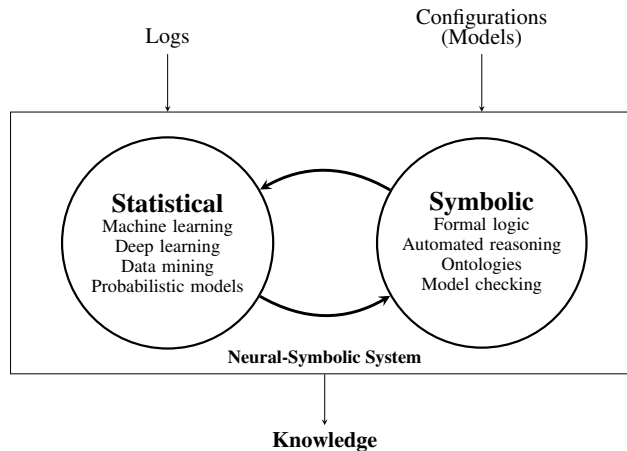

Figure 2: Revisit of figure from Hoehndorf *et al.* (2017)

entities of the logs. Then, it combines these vectors to represent log entry entities, using these representations to create visualizations. A powerful query language enables users to query the entire system for positive examples, by statically searching the logs or by computing correlations on the vector embeddings, and it uses formal engines to generate negative examples or even positive examples, when the query is only partially expressed or expressed as complex set of constraints. In this case, the biased insights from observational data are complemented by queries using the formal engine, providing users with a comprehensive insights that enables one to draw conclusions. The rest of the paper is organized as follows: Section 2 outlines the related work. In section 3 we will introduce the necessary background on Representation learning using embedding and its applications. Section 3 also introduces Formal methods on security policy. In section 4, we present our approach. Finally, in section 5, we conclude the paper by presenting our experimental evaluation on a real data and outlining the areas for future research.

2 Related Work

Machine learning has proved very powerful but has the limitation of interpretability. Symbolic AI was the main direction taken by AI research for a long time. Neuro-symbolic AI is a relatively new approach to AI that combines machine learning and symbolic AI. Neuro-symbolic AI can be seen as largely working within the framework of symbolic AI, using machine learning to infer and build the formal model upon which symbolic AI can reason Garcez *et al.* (2008). Our approach differs from Neuro-symbolic AI in that we begin with a formal model, and its observational data, and move forward from there. Rather than build a formal model from the ML model, our approach treats the ML model and formal model as collaborative players in a framework that accommodates complex real-world rule sets to enhance our understanding of the data flows they govern.

The formal model studied in this paper begins with firewall configuration components and policy rules. These rules are built by people over time. They grow and accrete as their underlying networks concerns change. Maintenance of policy sets is challenging given network complexity and the changing landscape of threats. Of high importance are questions such as: do these rules keep my network secure? Is a particular rule letting in unwanted traffic? Are my rules too strict, and disallowing necessary traffic? Are the rules creating bottlenecks? Have route changes or network changes rendered my rules obsolete? Bodei et. al have developed methods to assess firewall configurations using a formal model Bodei *et al.* (2018). In Log2NS a machine learned model of the observational data and a formal model of the firewall configuration work together to increase understanding of possible network problems and potential flaws or inconsistencies in the firewall configuration.

3 Background

3.1 Representation Learning using Embedding

The problem of analyzing logs for fault detection and diagnosis, particularly in security applications, is not new and has been a long rich area of research Du *et al.* (2017); Fu *et al.* (2009); Ring *et al.* (2019); Lopez-Martin *et al.* (2017); Ibrahim *et al.* (2016); Xu *et al.* (2009); Reidemeister *et al.* (2011). To best automate log analysis using ML techniques, the first step is effective log representation. This section reviews a series of recently introduced methodologies from the field of representation learning that attempt to learn from complex high dimensional space by first transforming them into vectors, followed by a downstream learning task, e.g., say calculating behavioral similarities between network logs.

Word2Vec is arguably the most popular word embedding model proposed by Mikolov et al Mikolov *et al.* (2013a) that project words to vectors of real numbers. Mikolov et al. proposed two different architectures; 1). Continuous bag of words (CBOW) neural network that maximizes the probability of a word given its context as in Eq. 1, and 2) Skip-gram model which uses a current word to predict the context words as in Eq. 2.

$$\frac{1}{\eta} \sum_{n=1}^{\eta} \log p(w_n | w_{n-\frac{c}{2}} ... w_{n+\frac{c}{2}}) \tag{1}$$

$$\frac{1}{\eta} \sum_{n=1}^{\eta} \sum_{m=n-c, m \neq n}^{n+c} \log p(w_m | w_n) \tag{2}$$

where η is vocabulary size and c is the size of context for each word. This also inspired some well known successful adaptation of Word2vec for nodes in large graph like Deepwalk Perozzi *et al.* (2014) and its extension node2vec Grover & Leskovec (2016). However, these methods suffer due to the inability to generalize to unseen data instances or to encode node attributes for graph embedding. Consequently, Hazem et al proposed a Graph Neural networks (GNNs) based method to perform graph representation learning at scale with better generalization Soliman *et al.* (2020). The node embeddings are calculated using an information diffusion mechanism, where nodes broadcast information to their neighbours until convergence. With the advent of GNN, several graph-based representation techniques have emerged Hamilton *et al.* (2017); Peng *et al.* (2018). The most representative work in this line is Text-GCN by Yao et al. that achieves state-of-the-art results on benchmark domains Yao *et al.* (2019) with one big disadvantage of high memory utilization. Pennington et al. proposed GloVe, a method that combines the benefits of skip-gram and global matrix

factorization. GloVe is based on the word-context co-occurrence matrix as:

$$\mathcal{J} = f(\#(w_n, c_m))(\mathbf{w}_n^T \mathbf{c}_m + b_{w_n} + b_{c_m} - \log \#(w_n, c_m))^2$$

where b_{w_n}, b_{c_m} are scalar biases for the target and context words. Ring et al. extended Word2vec and proposed IP2Vec that aims at transforming IP addresses into a continuous feature space $\mathbb{R}^m$ to compute distances between the IP addresses in this feature space Ring *et al.* (2017). They employ simple traffic descriptors as the context to train the Word2Vec model. Their empirical evaluations demonstrate the superiority of IP2Vec over other embedding techniques techniques within a botnet data set. In this paper, we will employ similar embeddings to learn meaningful vector representations of the network logs.

3.2 Applications of Embedding

Analysis and clustering of network traffic logs aim at answering several interesting questions in the domain of cybersecurity such as identifying host behaviors, grouping hosts with similar intentions, and so on. Unfortunately, this is an arduous task for humans to classify these logs that are generated in large volumes. Consequently, more often than not, this problem is posed as an unsupervised learning problem. Many approaches have been proposed to effectively condense or summarize these logs. These include, anomaly or outlier detection Münz *et al.* (2007); Zhong & Khoshgoftaar (2007); Wang *et al.* (2017, 2013), network traffic classification Pujari *et al.* (2017); Erman *et al.* (2006); Glennan *et al.* (2016); Singh *et al.* (2016), and data security applications like policy generation, signature detection etc. Finamore *et al.* (2011), to name a few.

Earlier systems heavily relied on rule-based approaches to classify or group log entries. These rules are limited to specific application scenarios and require extensive domain knowledge making the problem harder Cinque *et al.* (2012); Prewett (2003); Rouillard (2004). Other generic methods typically apply a two-step procedure; first, parse log entries to structured forms (following the methodologies listed in section 3.1) and then employ the learning task say clustering. A clustering function c takes a set of feature vectors as inputs and allocates them to appropriate clusters based on a similarity measure such that $c : \mathbb{R}^{|N|xd} \implies \mathbb{N}^N$. Choosing a clustering algorithm from the many options like Kmeans MacQueen *et al.* (1967), Gaussian Mixture Models Bishop (2006) etc. for the use case in hand is a non-trivial task.

McGregor et al. proposed AutoClass, a probabilistic model-based clustering analysis to group logs using features from the transport layer McGregor *et al.* (2004). Zander et al. proposed an enhancement of the Bayesian clustering technique using an expectation–maximization (EM) algorithm that helps determine the number of clusters as well as supports the soft clustering of the data. Zander *et al.* (2005). However, the EM algorithm is often rather slow. In this work, we use the most efficient partition-based clustering method K-means which is a simple yet fast algorithm. For the set of feature vectors $\mathbb{X} \subseteq \mathbb{R}^d$, the k-means objective is to find a set $C = c_1, ..c_k$ of $kclusters$, such that it minimizes the average squared error $SSE = \sum_{x \in \mathbb{X}} min_{c \in C}(dist(x, c))$. Another successful method, DBSCAN, forms clusters based on the notion of density-reachability, i.e. A point is directly density-reachable to the objects in ϵ-neighborhood of this point. However, the approach itself is very sensitive to the parameters. Erman et al. compared empirically the effectiveness of K-means, DBSCAN, and AutoClass clustering for the traffic analysis and classification task, wherein the authors demonstrate the superiority of K-means over other state-of-the-art methods in learning high-purity clusters Erman *et al.* (2006).

3.3 Formal methods on security policy

Formal methods encompass a group of technologies for mathematical reasoning about the sanity of system behaviors. They have found successful applications in specifying, building and verifying complex software and hardware systems D'silva *et al.* (2008); Stewart *et al.* (2021); Bernardi *et al.* (2021); Biere (2021). A formal model of a given system is a precise mathematical description of its components and their relationships Edwards *et al.* (1997). Taken together, the model represents system behavior. Formal models are usually stated via mathematical formulae, often equations Barrett *et al.* (2010). Formal engines De Moura & Bjørner (2008) perform satisfiability checks to determine if the behavior of the system satisfies a given property, which also is described using a formal representation. A property in this context is a high level description of system behavior. If a solution exists, a trace depicting the solution is generated, otherwise engines return unsatisfiable i.e. no solution exists.

Formal synthesis/verification have proven to be effective in validation of firewall security policies Liu (2008); Bodei *et al.* (2018); Beckett *et al.* (2017). A firewall configuration is composed of logical components, whose interactions define the firewall behavior. Security rules within the policy specify which traffic to accept or reject, based on filtering conditions. Policy administrators make use of a variety of filters such as IP addresses, geographical zones, applications etc. to specify security rules. Further, the specification enforces a precedence order amongst the rules. Examples of logical components in a configuration:

- network interfaces , zones, addresses, address groups
- applications, application groups
- address translation rules (NAT)
- user, security profiles
- routing tables
- security rules
- address, address groups

To build a formal model from a firewall configuration, all its logical components and their relationships are converted to formulas.

4 Our Approach

Algorithm 1 depicts the steps involved in Log2NS approach. It consists of two collaborative models, with a unified query interface for user interactions. As shown in Fig. 3, the query engine routes user queries and interacts with the two models to return a final set of results. The Formal model is built by first extracting logical components from input firewall configuration files. Thereafter each logical component is converted to an equivalent formal representation. For example, Table 1 shows a security rule (SR1) in firewall configuration and its conversion that allows traffic from *Trust zone* to *Internal zone* via any application.

Table 1: An example security rule

Rule name	From Zone	Application	To Zone	Action
SR1	Trust	Any	Internal	Allow

$$\text{From\_Zone} \in ['Trust'] \wedge \text{To\_Zone} \in ['Internal'] \rightarrow \text{Action} \in [1] \tag{3}$$

$$\text{Zone} \in ['Trust', 'Untrust', 'Internal'] \tag{4}$$

$$\text{Action} \in [0, 1] \tag{5}$$

Entities like Source IP, Dest IP, Applications, Actions etc are considered as independent variables (Eq. 3). Some of these variables (as shown in Eq. 4, 5) can take values from a discrete set of possible values. The discrete set is inferred from the configuration. To provide solutions to user queries, an open source network configuration analysis tool is used Fogel *et al.* (2015).

Firewall traffic logs are the other set of inputs to Log2NS for building the machine learning model. Traffic logs consist of a defined set of entities $e_1, e_2, \ldots e_n$; example - *Source IP address, Destination IP address, IP protocol, Source port, Destination port* and *Bytes sent*. Depending on the device and vendor these flow logs may consist of additional parameters such as *From zone, To zone* and *Application*. Many of these entities are categorical and do not carry natural order; thus their meaning depends on context. To analyze logs we first create a global embedding of these entities via an approach similar to IP2Vec. Each log entry can be seen as a sentence as it carries meaning. For example, *[10.0.0.1, 8.8.8.8, TCP, 80, 100]* states that a local machine is talking to Google server over port 80 using the TCP protocol and has sent 100 bytes of data. Log2NS defines context and target pairs from the set of log entities. Once the context-target pairs are generated for the input logs, the entity embedding gets computed by training a fully connected neural network with a single hidden layer whose size is much smaller than the input and output. The input of this
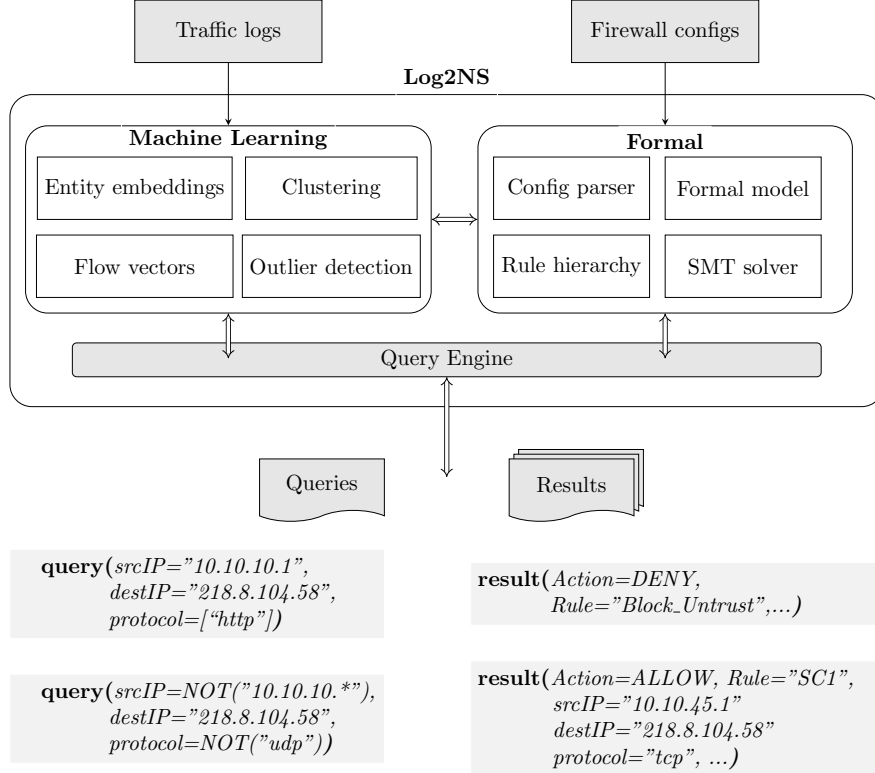
Figure 3: Log2NS architecture diagram

network is a one-hot encoded context vector while the output is the probability of the target vector computed via a softmax function. The weights that map the input to the hidden layer give us the entity embedding. In practice, computing the probability of the target vectors becomes intractable due to the denominator of the softmax computation. There are two ways to handle this. One is to use negative sampling Mikolov *et al.* (2013a), wherein the training step, instead of iterating over the entire vocabulary, we generate several negative examples from the noise distribution of the words. Because these negative examples have lower probability in the corpus, they simplify the softmax computation. Another approach is to use hierarchical softmax that uses an efficient binary tree format to represent the words in the vocabulary Mikolov *et al.* (2013b). We use hierarchical softmax to compute the embedding for two reasons: 1. it works better for a corpus with infrequent words and in our case, some of the entity vectors might be present in only a few logs; 2. while generating the context-target pairs we ignore some of the possible combinations that appear in the same row (one observation logged from the firewall) as they should have zero impact on the embedding. Negative sampling might generate a few context-target pairs within the same row and assigning negative impact to these pairs might cause inaccurate embedding.

Once the embeddings for the all the entities $e_1, e_2, \ldots e_n$ are computed using the above procedure, we generate a vector for each row by concatenating the entity vectors. However, the concatenated vector might contain correlated entries. To remove the correlation and obtain a more compact low-dimensional representation, one can use non-linear manifold learning techniques such as kernel principal component analysis (KPCA) Schölkopf *et al.* (1997) and diffusion maps Coifman & Lafon (2006). Often times, using the correct distance metric to compute the correlation becomes important. Some of the manifold learning techniques such as diffusion maps allows us to measure intrinsic distance between the vectors Thimmisetty *et al.* (2014, 2018). Instead of concatenation, one can also use weighted average to assign importance to some of the entities. Once the vectors for each row is computed, we use unsupervised clustering techniques (K-means and Gaussian mixture models) to identify patterns in the logs.

Algorithm 1: Log2NS algorithm

Input: logs, configuration_file
Output: patterns_in_logs, positive_examples, negative_examples
1 Generate context and target pairs
2 Compute the embedding of the log entities with hierarchical softmax
3 Generate vector for each row of the log entry
4 Create clusters using unsupervised learning
5 Create a formal model with configuration_file
6 Identify the key question to be addressed
7 Use formal reasoning and log querying to generate positive and negative examples

5   Experimental Evaluation
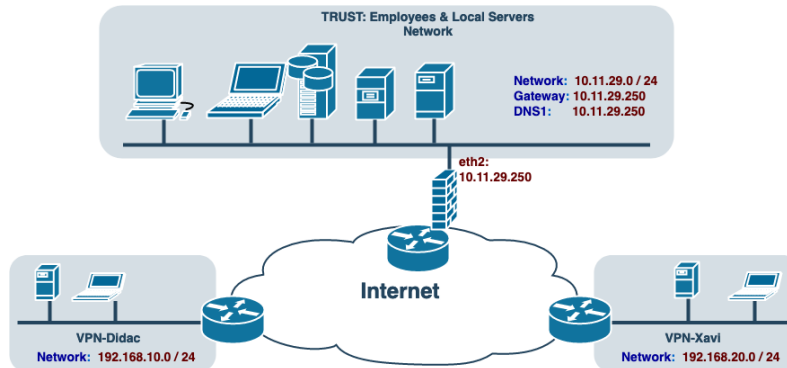
5.1   Data Description

Figure 4: Network topology

In our experiments, we worked with network traffic data[3] from a real network with the topology shown in Fig. 4. The network has three sub networks of interest: a firewall-protected main office and two remote offices. The office firewall has IP address 10.11.29.250. The observational data spanned a period of three weeks. Raw flow records included source and destination IP addresses and ports, protocol, utilization, source and destination zones, identified applications, TCP flag and tunnel information. The network traffic is generated by users and by devices both inside and outside the network. Firewall configuration files during this period were also stored.

5.2   Experimental Setup

We demonstrate the Log2NS framework by building ML model from the observational data, symbolic model from the firewall configuration. We show the effectiveness of our approach in analyzing the access patterns and associated security rules.

5.2.1   Building the Log Data Model

The statistical model was created using three weeks of traffic data. Traffic log entries consisted of: *Source Address, Destination Address, Application, IP Protocol, Source Region, Destination Region.* To generate embeddings, the context target pairs (Table 2) were used. We used the embedding approach described earlier and implemented using the Gensim package Rehurek *et al.* (2011). We grouped the data into 20 clusters using K-Means Pedregosa *et al.* (2011) and the model hyper parameters were obtained by grid search.

---

[3]Dataset will be provided as a supplemental material with Apache 2.0 licence

8

Table 2: Context-Target Pairs

Context	Targets
Source Address	Destination Address, Application
Application	Destination Address
Destination Region	Source Address, Application

5.2.2 Building the Formal Model

To build the formal model, a previously generated snapshot (same period as the traffic logs) of firewall configuration files were provided as inputs to Log2NS.

5.2.3 Generating Questions and Reasoning about our Models

The next phase of the Log2NS framework is an iterative process: observations from the statistical model motivate queries to the symbolic model, which can lead to further observations and questions. From the clusters generated by the statistical model, we picked following positive examples of behavior to investigate: traffic to an unintended region, and dns queries outside of the firewall.

a) Traffic to an unintended region was seen in the following cluster:

Table 3: Unintended Region Cluster

Source IPs	Destination IPs	Applications
10.11.29.5	42.62.94.2	
six additional	twenty-two additional	not-applicable
source IPs	destination IPs	

This cluster stood out as its showed traffic to an unintended region. Further there was no application identified for the traffic. Through the unified query engine, Log2NS was able to confirm that this traffic is (and will be) allowed only by BypassFw security rule (Table 4). This exposed mis-configurations in policy, which could lead to security threats.

Table 4: Symbolic Model answers to Access to undesired region

Filter Name	Action	Line	Trace
zone Trust vsys1 to zone Untrust vsys1	PERMIT	BypassFW	- Matched security rule BypassFW - Matched source address - Matched address any - Matched destination address - Matched service application-default - Matched application any

b) The next cluster of interest (Table 5) showed source IP addresses that connected to internet dns servers (4.4.4.4 and 8.8.8.8).

Table 5: DNS Queries to Internet Cluster

Source IPs	Destination IPs	Applications
192.168.1.254	4.4.4.4	
10.11.29.222	8.8.8.8	dns
10.11.29.6		

We wanted to disable access to internet dns servers, such that dns requests use local dns server (10.11.29.250). We added new security rules to disallow such accesses and validated the change with Log2NS.

5.3 Future directions

Building an accurate formal model from a specified firewall configuration is often the most critical and challenging step towards using such a model for property verification. The Log2NS framework will be enhanced to select a fixed number of traffic logs pertaining to security rules, to create formal 'witness' properties. Witness properties check for the existence of specified traffic scenarios against the formal model. Any failures will point to an over-constrained model, indicating nuances not captured during formal model creation. Further, Log2NS will be enhanced to perform incremental training of the machine learning model. The traces generated by the formal model would be used as pseudo input logs for training.

6 Conclusion

In this paper, we addressed the problem of enhancing insights from observational data coming from logs by using formal engines. We showed use cases, where logs can suffer from the survivorship bias problem, and because of that, insights derived from them will be inherently biased, if analysis is not complemented by other methods.

We introduced Log2NS, a framework that enables users to reason about logs and to understand a complex system by using deep learning techniques on logs. We captured system behavior using embeddings on log entries, that were later combined, and clustering techniques were used to understand interesting scenarios. Because log data is biased, we enhanced the logs using formal technology. Formal engines and a constraint language that accepted partially specified constraints and negation enabled a user to query the framework to understand rare conditions or negative examples. We showed how the framework was used to extract insights in a network security environment, where logs were obtained from firewall network and security configuration. A corresponding formal model was used to determine if traffic could be accepted or denied, and explore optimizations of security rules.

References

Barrett, Clark, Stump, Aaron, Tinelli, Cesare, *et al.* 2010. The smt-lib standard: Version 2.0. *Page 14 of: Proceedings of the 8th international workshop on satisfiability modulo theories (Edinburgh, England)*, vol. 13.

Beckett, Ryan, Gupta, Aarti, Mahajan, Ratul, & Walker, David. 2017. A general approach to network configuration verification. *Pages 155–168 of: Proceedings of the Conference of the ACM Special Interest Group on Data Communication.*

Bernardi, S, Gentile, U, Marrone, S, Merseguer, J, & Nardone, R. 2021. Security modelling and formal verification of survivability properties: Application to cyber–physical systems. *Journal of Systems and Software*, **171**, 110746.

Biere, Armin. 2021. Bounded model checking. *Pages 739–764 of: handbook of Satisfiability*. IOS Press.

Bishop, Christopher M. 2006. Mixture models and the EM algorithm. *Microsoft Research, Cambridge*.

Bodei, Chiara, Degano, Pierpaolo, Galletta, Letterio, Focardi, Riccardo, Tempesta, Mauro, & Veronese, Lorenzo. 2018. Language-independent synthesis of firewall policies. *Pages 92–106 of: 2018 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE.

BOTTYÁN, ZSOLT, & PALIK, MÁTYÁS. 2010. The accident of AF-447 flight–analysis and reconstruction of weather situation along the flight path. *Technology*, **9**(2), 217–227.

Calabrese, Matteo, Cimmino, Martin, Fiume, Francesca, Manfrin, Martina, Romeo, Luca, Ceccacci, Silvia, Paolanti, Marina, Toscano, Giuseppe, Ciandrini, Giovanni, Carrotta, Alberto, *et al.* 2020. SOPHIA: An event-based IoT and machine learning architecture for predictive maintenance in industry 4.0. *Information*, **11**(4), 202.

Cinque, Marcello, Cotroneo, Domenico, & Pecchia, Antonio. 2012. Event logs for the analysis of software failures: A rule-based approach. *IEEE Transactions on Software Engineering*, **39**(6), 806–821.

Clarke, Edmund, McMillan, K, Campos, Sérgio, & Hartonas-Garmhausen, Vasiliki. 1996. Symbolic model checking. *Pages 419–422 of: International conference on computer aided verification*. Springer.

Coifman, Ronald R, & Lafon, Stéphane. 2006. Diffusion maps. *Applied and computational harmonic analysis*, **21**(1), 5–30.

Dangut, Maren David, Skaf, Zakwan, & Jennions, Ian K. 2020. An integrated machine learning model for aircraft components rare failure prognostics with log-based dataset. *ISA transactions*.

De Moura, Leonardo, & Bjørner, Nikolaj. 2008. Z3: An efficient SMT solver. *Pages 337–340 of: International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer.

D'silva, Vijay, Kroening, Daniel, & Weissenbacher, Georg. 2008. A survey of automated techniques for formal software verification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, **27**(7), 1165–1178.

Du, Min, Li, Feifei, Zheng, Guineng, & Srikumar, Vivek. 2017. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. *Pages 1285–1298 of: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*.

Edwards, Stephen, Lavagno, Luciano, Lee, Edward A, & Sangiovanni-Vincentelli, Alberto. 1997. Design of embedded systems: Formal models, validation, and synthesis. *Proceedings of the IEEE*, **85**(3), 366–390.

Erman, Jeffrey, Arlitt, Martin, & Mahanti, Anirban. 2006. Traffic classification using clustering algorithms. *Proceedings of the 2006 SIGCOMM Workshop on Mining Network Data, MineNet'06*, **2006**(January), 281–286.

Fang, Weijian, Tan, Xiaoling, & Wilbur, Dominic. 2020. Application of intrusion detection technology in network safety based on machine learning. *Safety Science*, **124**, 104604.

Finamore, Alessandro, Mellia, Marco, & Meo, Michela. 2011. Mining unclassified traffic using automatic clustering techniques. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, **6613 LNCS**, 150–163.

Fogel, Ari, Fung, Stanley, Pedrosa, Luis, Walraed-Sullivan, Meg, Govindan, Ramesh, Mahajan, Ratul, & Millstein, Todd. 2015. A general approach to network configuration analysis. *Pages 469–483 of: 12th {USENIX} symposium on networked systems design and implementation ({NSDI} 15)*.

Fontaine, Jaron, Kappler, Chris, Shahid, Adnan, & De Poorter, Eli. 2019. Log-based intrusion detection for cloud web applications using machine learning. *Pages 197–210 of: International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*. Springer.

Fu, Qiang, Lou, Jian-Guang, Wang, Yi, & Li, Jiang. 2009. Execution anomaly detection in distributed systems through unstructured log analysis. *Pages 149–158 of: 2009 ninth IEEE international conference on data mining*. IEEE.

Garcez, Artur S. d'Avila, Lamb, Luis C., & Gabbay, Dov M. 2008. *Neural-Symbolic Cognitive Reasoning*. 1 edn. Springer Publishing Company, Incorporated.

Glennan, Timothy, Leckie, Christopher, & Erfani, Sarah M. 2016. Improved classification of known and unknown network traffic flows using semi-supervised machine learning. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, **9723**, 493–501.

Grover, Aditya, & Leskovec, Jure. 2016. node2vec: Scalable feature learning for networks. *Pages 855–864 of: Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*.

Gupta, Aarti. 1992. Formal hardware verification methods: A survey. *Formal Methods in System Design*, **1**(2-3), 151–238.

Haixiang, Guo, Yijing, Li, Shang, Jennifer, Mingyun, Gu, Yuanyue, Huang, & Bing, Gong. 2017. Learning from class-imbalanced data: Review of methods and applications. *Expert Systems with Applications*, **73**, 220–239.

Hamilton, William L, Ying, Rex, & Leskovec, Jure. 2017. Inductive representation learning on large graphs. *arXiv preprint arXiv:1706.02216*.

Harel, David, & Pnueli, Amir. 1985. On the development of reactive systems. *Pages 477–498 of: Logics and models of concurrent systems*. Springer.

Hoehndorf, Robert, Queralt-Rosinach, Núria, *et al.* 2017. Data science and symbolic AI: Synergies, challenges and opportunities. *Data Science*, **1**(1-2), 27–38.

Huang, Qijing, Shojaei, Hamid, Zyda, Fred, Nazi, Azade, Vasudevan, Shobha, Chatterjee, Sat, & Ho, Richard. 2021. Faster Coverage Convergence with Automatic Test Parameter Tuning in Constrained Random Verification.

Ibrahim, Hamza Awad Hamza, Aqeel Al Zuobi, Omer Radhi, Al-Namari, Marwan A., MohamedAli, Gaafer, & Abdalla, Ali Ahmed Alfaki. 2016. Internet traffic classification using machine learning approach: Datasets validation issues. *Pages 158–166 of: 2016 Conference of Basic Sciences and Engineering Studies (SGCAC)*.

Kropf, Thomas. 2013. *Introduction to formal hardware verification*. Springer Science & Business Media.

Lamb, Luis, Garcez, Artur, Gori, Marco, Prates, Marcelo, Avelar, Pedro, & Vardi, Moshe. 2020. Graph neural networks meet neural-symbolic computing: A survey and perspective. *arXiv preprint arXiv:2003.00330*.

Liu, Alex X. 2008. Formal verification of firewall policies. *Pages 1494–1498 of: 2008 IEEE International Conference on Communications*. IEEE.

Lopez-Martin, Manuel, Carro, Belen, Sanchez-Esguevillas, Antonio, & Lloret, Jaime. 2017. Network traffic classifier with convolutional and recurrent neural networks for Internet of Things. *IEEE Access*, **5**, 18042–18050.

MacQueen, James, *et al.* 1967. Some methods for classification and analysis of multivariate observations. *Pages 281–297 of: Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, vol. 1. Oakland, CA, USA.

McGregor, Anthony, Hall, Mark, Lorier, Perry, & Brunskill, James. 2004. Flow clustering using machine learning techniques. *Pages 205–214 of: International workshop on passive and active network measurement*. Springer.

Mehta, Ashok B. 2020. *SystemVerilog Assertions and Functional Coverage*. Springer.

Mikolov, Tomas, Sutskever, Ilya, Chen, Kai, Corrado, Greg, & Dean, Jeffrey. 2013a. Distributed representations of words and phrases and their compositionality. *arXiv preprint arXiv:1310.4546*.

Mikolov, Tomas, Chen, Kai, Corrado, Greg, & Dean, Jeffrey. 2013b. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Münz, Gerhard, Li, Sa, & Carle, Georg. 2007. Traffic Anomaly Detection Using K-Means Clustering. *GI/ITG Workshop MMBnet*, 13—-14.

Nikolenko, Sergey I. 2019. Synthetic data for deep learning. *arXiv preprint arXiv:1909.11512*.

Omar, Salima, Ngadi, Asri, & Jebur, Hamid H. 2013. Machine learning techniques for anomaly detection: an overview. *International Journal of Computer Applications*, **79**(2).

Pedregosa, Fabian, Varoquaux, Gaël, Gramfort, Alexandre, Michel, Vincent, Thirion, Bertrand, Grisel, Olivier, Blondel, Mathieu, Prettenhofer, Peter, Weiss, Ron, Dubourg, Vincent, *et al.* 2011. Scikit-learn: Machine learning in Python. *the Journal of machine Learning research*, **12**, 2825–2830.

Peng, Hao, Li, Jianxin, He, Yu, Liu, Yaopeng, Bao, Mengjiao, Wang, Lihong, Song, Yangqiu, & Yang, Qiang. 2018. Large-scale hierarchical text classification with recursively regularized deep graph-cnn. *Pages 1063–1072 of: Proceedings of the 2018 world wide web conference.*

Perozzi, Bryan, Al-Rfou, Rami, & Skiena, Steven. 2014. Deepwalk: Online learning of social representations. *Pages 701–710 of: Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining.*

Pratt, Vaughan. 1995. Anatomy of the Pentium bug. *Pages 97–107 of: Colloquium on Trees in Algebra and Programming.* Springer.

Prewett, James E. 2003. Analyzing cluster log files using logsurfer. *In: Proceedings of the 4th Annual Conference on Linux Clusters.* Citeseer.

Pujari, Triveni, Nalinakshi, B G, & Jayaramaiah, D. 2017. Flow-Based Network Traffic Classification Using Clustering Technique with MLA. **5**(Vii), 1855–1862.

Rehurek, Radim, Sojka, Petr, *et al.* 2011. Gensim—statistical semantics in python. *Retrieved from genism. org.*

Reidemeister, Thomas, Jiang, Miao, & Ward, Paul A.S. 2011. Mining unstructured log files for recurrent fault diagnosis. *Pages 377–384 of: 12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops.*

Ring, Markus, Dallmann, Alexander, Landes, Dieter, & Hotho, Andreas. 2017. Ip2vec: Learning similarities between ip addresses. *Pages 657–666 of: 2017 IEEE International Conference on Data Mining Workshops (ICDMW).* IEEE.

Ring, Markus, Schlör, Daniel, Landes, Dieter, & Hotho, Andreas. 2019. Flow-based network traffic generation using generative adversarial networks. *Computers & Security*, **82**, 156–172.

Rouillard, John P. 2004. Real-time Log File Analysis Using the Simple Event Correlator (SEC). *Pages 133–150 of: LISA*, vol. 4.

Schölkopf, Bernhard, Smola, Alexander, & Müller, Klaus-Robert. 1997. Kernel principal component analysis. *Pages 583–588 of: International conference on artificial neural networks.* Springer.

Shermer, Michael. 2014. How the survivor bias distorts reality. *Scientific American*, **1**, 106.

Singh, M. P., Srivastava, Gargi, & Kumar, Prabhat. 2016. Internet traffic classification using machine learning. *International Journal of Database Theory and Application*, **9**(12), 45–54.

Soliman, Hazem M, Salmon, Geoff, Sovilij, Dusan, & Rao, Mohan. 2020. A Graph Neural Network Approach for Scalable and Dynamic IP Similarity in Enterprise Networks. *arXiv preprint arXiv:2010.04777.*

Stewart, Danielle, Liu, Jing Janet, Cofer, Darren, Heimdahl, Mats, Whalen, Michael W, & Peterson, Michael. 2021. AADL-based safety analysis using formal methods applied to aircraft digital systems. *Reliability Engineering & System Safety*, **213**, 107649.

Sutton, Richard S, & Barto, Andrew G. 2011. *Reinforcement learning: An introduction.*

Takakis, Zacharias, Mangiras, Dimitrios, Nicopoulos, Chrysostomos, & Dimitrakopoulos, Giorgos. 2019. Dynamic Adjustment of Test-Sequence Duration for Increasing the Functional Coverage. *Pages 61–66 of: 2019 IEEE 4th International Verification and Security Workshop (IVSW).* IEEE.

Thimmisetty, Charanraj, Khodabakhshnejad, Arman, Jabbari, Nima, Aminzadeh, Fred, Ghanem, Roger, Rose, Kelly, Bauer, Jennifer, & Disenhof, Corinne. 2014. Multiscale stochastic representation in high-dimensional data using Gaussian processes with implicit diffusion metrics. *Pages 157–166 of: International Conference on Dynamic Data-Driven Environmental Systems Science.* Springer.

Thimmisetty, Charanraj A, Ghanem, Roger G, White, Joshua A, & Chen, Xiao. 2018. High-dimensional intrinsic interpolation using Gaussian process regression and diffusion maps. *Mathematical Geosciences*, **50**(1), 77–96.

Wald, Abraham. 1980. *A Reprint of 'A Method of Estimating Plane Vulnerability Based on Damage of Survivors*. Tech. rept. CENTER FOR NAVAL ANALYSES ALEXANDRIA VA OPERATIONS EVALUATION GROUP.

Wang, Jie, Yang, Lili, Wu, Jie, & Abawajy, Jemal H. 2017. Clustering analysis for malicious network traffic. *IEEE International Conference on Communications*, 2–7.

Wang, Jing, Rossell, Daniel, Cassandras, Christos G., & Paschalidis, Ioannis Ch. 2013. Network anomaly detection: A survey and comparative analysis of stochastic and deterministic methods. *Proceedings of the IEEE Conference on Decision and Control*, 182–187.

Xu, Wei, Huang, Ling, Fox, Armando, Patterson, David, & Jordan, Michael I. 2009. Detecting Large-Scale System Problems by Mining Console Logs. SOSP '09. Association for Computing Machinery.

Yao, Liang, Mao, Chengsheng, & Luo, Yuan. 2019. Graph convolutional networks for text classification. *Pages 7370–7377 of: Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33.

Zander, Sebastian, Nguyen, Thuy, & Armitage, Grenville. 2005. Automated traffic classification and application identification using machine learning. *Pages 250–257 of: The IEEE Conference on Local Computer Networks 30th Anniversary (LCN'05) l*. IEEE.

Zhong, S H I, & Khoshgoftaar, Taghi M. 2007. Clustering-based network intrusion detection. **14**(2), 169–187.