Log and Execution Trace Analytics System

Nazrin Abbasli [1], Murat Can Ganiz [1,2]
[1]Department of Computer Engineering, Marmara University
[2]VeriUs Teknoloji
Istanbul/Turkey
nazrinabbasli@marun.edu.tr
murat.ganiz@marmara.edu.tr

***Abstract*—Log files are available on every computer system. They automatically record important run time events of operating systems or software applications. They are mainly used to find the root cause of failures. Analyzing such log files allows us to detect anomalies, problems and improve the system. Since the log files are usually unstructured or semi-structured, the important task of log analysis is to parse usually immense amount of log message strings into the human readable and actionable reports. In this paper, we propose an implementation of a machine learning based log parsing system using Named Entity Recognition which is the process of identifying and categorizing entities in the text. Our approach make use of Bidirectional Encoder Representations from Transformers (BERT) to extract entities. The paper reports the results of experiments on two benchmark; macOS and Linux OS datasets.**

Keywords - Log Parsing, Log Analysing, NER, BERT, Drain, Tagtog

I. Introduction

Log files contain records of system states and events from operating systems and software. Such log data is available on practically every computer system and is a valuable resource for monitoring the system status, investigating system failures, anomaly detection, and henceforth cybersecurity. Analyzing logs for different purposes in an effective way is challenging. First, huge amounts of logs are generated routinely by software systems, so it is usually impractical to manually inspect log messages, for example for situational awareness. Second, logs related to errors or anomalies usually constitutes only a very small fraction of the data which makes training of the supervised machine learning systems difficult. In addition, these errors and anomalies can be quite system specific, which makes generalizations to other systems and domains difficult. Third, logs are usually in semi-structured form, they need to be converted to structured form and their format can vary from system to system.

Generally, system logs are a composition of constant strings and variable values. To enable log analysis, the first and essential step is log parsing. Event log parsing is a process to split and label each field in a log entry such that ”timestamp”, ”hostname”, ”IP address”, ”level”, ”server”, ”PID”, and ”log message”. Log parsing can be categorized as rule-based, data-driven parsing, and source code-based. Most current log management tools support rule-based parsing. Since log file formats differ, defining the set of rules for a new system is time-consuming. Furthermore, the static rules can not be easily updated or extended when version changes of the same software happens.

In order to relax the inherent weaknesses of rule based systems, in this study, we model log parsing as the Named Entity Recognition problem (NER) and use Bidirectional Encoder Representations from Transformers (BERT) to extract entities for efficient, dynamic and generalizable parsing of the logs.

The organization of this paper is as follows. Section II summarizes the background and related work. Approach for modeling the log parsing as NER is provided in Section III. Section IV includes detailed information about the experimental setup, and the results and discussion can be found in Section V. Finally, the paper concludes in Section VI.

II. Background And Related Work

Logs are important in the development and maintenance of computer systems. Log analysis can be used for:

Usage analysis: User behaviour analysis, API profiling, workload modeling can be listed as usage analysis applications and requires structured events as the input.

Anomaly detection is an essential task to construct a trust-worthy computer system. Log parsing is required by machine learning algorithms such as Principal Component Analysis (PCA), Invariant Mining (IM), and Deep Learning (DL) that use log entries for anomaly detection.

Performance modeling: Facebook has recently reported a use case to apply logs to performance modeling. The inputs of the performance model are structured event sequences.

Duplicate issue identification: System issues are re-

ported repeatedly by different users, which leads to duplicate problems. Structured data is required to automatically identify duplicates.
Failure diagnosis: Logs are enormous in volume and can be quite messy. So, manually diagnosing the system is time-consuming and challenging.

There are many studies using machine learning algorithms for anomaly detection [1]. We focus detection of anomalies in software logs. The recurrent neural network (RNN) language models are presented for anomaly log detection, and their performance is demonstrated on the Los Alamos National Laboratory (LANL) cybersecurity dataset [2]. LANL [3] consists of over one billion log lines collected over 58 consecutive days. The logs contain DNS, network flow, anonymized process, and authentication information.

Min Du et al. [4] have suggested DeepLog, a data-driven approach for anomaly detection. In this approach, log entries are viewed as elements of a sequence that follows specific patterns and grammar rules. DeepLog is a deep neural network that utilizes Long Short-Term Memory (LSTM) to model this log entry sequences [5]. Log entries are parsed into the log key and the parameter value vector for anomaly detection.

The standard evaluation metrics such as recall, precision, and F-score are used to evaluate log key anomaly detection methods of Principal Component Analysis (PCA), Invariant Mining (IM), N-gram, and Deeplog methods. Based on this comparison, DeepLog achieved the best overall performance, with an F-score of 96%. They note that PCA and IM are offline methods, so they cannot be perform anomaly detection online for incoming log entries.

Parsing unstructured log messages into a structured format provides efficient searching, filtering, grouping, counting, and log mining. There are certain methods used by the log parsers:
Frequent Pattern Mining - A frequent pattern is a set of frequently occurred items in a dataset. Examples of that approach are SLCT, LogCluster, and LFA. Clustering – LKE, LogMine, and LogSis are offline, SHISHO, LenMa are online methods that use the clustering method.
Heuristics – Log messages contain some unique characteristics. Drain, AEL, and IPLoM propose heuristics-based log parsing methods.

Loghub (Figure 1) data repository is freely available for research and contains 440 million log messages in size of 77 GB [6]. The evaluation of 13 log parsers on the loghub dataset is reported in terms of accuracy, robustness, and efficiency [7]. Drain automatic log parsing tool outperforms the existing online log parsing methods in terms of accuracy and efficiency.

Nerlogparser is an automatic event parsing tool that uses Bidirectional Long Short-Term Memory (BiLSTM)

Dataset	Description	Time Span	Data Size
			Distributed system logs
HDFS	Hadoop distributed file system log	38.7 hours	1.47 GB
Hadoop	Hadoop mapreduce job log	N.A.	48.61 MB
Spark	Spark job log	N.A.	2.75 GB
ZooKeeper	ZooKeeper service log	26.7 days	9.95 MB
OpenStack	OpenStack software log	N.A.	60.01 MB
			Supercomputer logs
BGL	Blue Gene/L supercomputer log	214.7 days	708.76 MB
HPC	High performance cluster log	N.A.	32.00 MB
Thunderbird	Thunderbird supercomputer log	244 days	29.60 GB
			Operating system logs
Windows	Windows event log	226.7 days	26.09 GB
Linux	Linux system log	263.9 days	2.25 MB
Mac	Mac OS log	7.0 days	16.09 MB
			Mobile system logs
Android	Android framework log	N.A.	3.38 GB
HealthApp	Health app log	10.5 days	22.44 MB
			Server application logs
Apache	Apache server error log	263.9 days	4.90 MB
OpenSSH	OpenSSH server log	28.4 days	70.02 MB
			Standalone software logs
Proxifier	Proxifier software log	N.A.	2.42 MB

Fig. 1: The Summary of LogHub Datasets (from [7])

[8].

Pokharel [9] found that extracting useful information from log messages is beneficial for real-time analysis and detecting faults, anomalies, and security threats. Thus, they proposed a model to extract information from the log messages. The approach is used to extract named entities by building classifiers based on Naïve Bayes (NB) and Support Vector Machines (SVM). Experiments were conducted on Windows OS and Exchange Mail Server logs. In comparison with the existing frequent item-set approach, they say they outperformed as the numbers of categories increase. They get 97% and 97.40% accuracy on Windows OS and Exchange Mail Server datasets respectively.

Named Entity Recognition (NER) is used in natural language processing (NLP) applications such as question answering, text understanding, and knowledge base construction [10]. NER labels sequence of words as a person, location, organization, etc [11]. There are four main streams of approaches to NER:

1. Rule-based approaches rely on hand-crafted rules designed based on domain-specific gazetteers [12], [13] and, syntactic-lexical patterns [14].

2. Unsupervised learning approaches - clustering is a typical approach to unsupervised learning [11]. Clustering-based NER systems employ context similarity to extract named entities from the clustered groups of terms.

3. Feature-based supervised learning approaches rely on supervised learning algorithms. NER is transformed into a multi-class classification or sequence labeling task.

Deep learning-based (DL-based) approaches automatically discover hidden representations required for the classification.

BiLSTM-CRF (Bidirectional LSTM-Conditional Random Fields) [15] is the most common architecture for DL-based NER and achieves state-of-the-art performance (93.5%) on CoNLL03. BERT achieves state-of-the-art performance (92.07%) on OntoNotes5.0.
CRF (Conditional Random Fields) is a powerful method when used with non-contextualized language model embeddings such as GloVe and Word2vec, but does not always perform better when used with contextual embeddings like ELMo and BERT [15], [16].

There are some key challenges in NER: 1) Supervised NER systems require big annotated data for training. That is a big challenge for the resource-poor languages; 2) The quality and consistency of the annotation are important because of the language ambiguity; 3) In many scenarios, NER deals with informal text and unseen entities which is more challenging because of nosiness and shortness.

ELK Stack is an abbreviation for three open-source projects: Elasticsearch, Logstash, and Kibana. ELK Stack is popular for log management and analysis. Elasticsearch is a search engine used to store, search large volumes of data. Logstash is a server-side data processing pipeline. Logstash allows us to collect data from different sources, transform, and send it to the desired destination. Kibana is an open-source tool for data visualization and exploration that used for log and time-series analytics and application monitoring.

Modern log management includes the following key abilities:
Aggregation – collecting and sending log data from multiple sources.
Processing – transforming log messages into easily understandable data.
Storage – Storage – storing data for monitoring, and analysis.
Analysis – creating visualizations and dashboards.

III. Approach

We model log parsing as a Named Entity Recognition (NER) problem and use BERT to extract important sections from log entries. We annotate a supervised dataset for log files, by using two methods: 1) using log entries in the output of the Drain log parser as tags; 2) manually tagging important sections using Tagtog annotation tool.

A. Challenges

NER systems require big annotated data. Log files, unlike ordinary text files, do not contain person, location, organization entities. Therefore, the well-known datasets and corpora for NER such as CoNLL 2003 [17], WNUT 17 Emerging Entities Dataset [18], Annotated GMB Corpus [19] cannot be used as the training datasets for log files. The other challenge is that BERT WordPiece tokenizer splits tokens into subwords and it is challenging to get the corresponding labels for those subwords.

Our approach to addressing the above challenges is described in this section.

B. Data Annotation

One of the most common tagging methods in NER tasks is BIO tagging, which is short for (Beginning Inside Other). Generally, entities are categorized into the "Person", "Organization", "Location" (for instance, B-Per, B-Loc, B-Org), etc. Since there is no tagged log dataset, so in our project, we are using custom tags such as *"Time", "User", "Component", "PID", "Level", "Content"*. Here *"Content"* is the log message which is necessary for log analysis. An example of custom tags for NER in the log entry is shown below:

Jun	9	06:06:20	combo	kernel:	zapping	low	mappings
Time	**Time**	**Time**	**level**	**component**	**O**	**O**	**O**

Fig. 2: Custom tags for NER in the log entry

Drain is an online, heuristic-based, and open-source log parsing method, and it is also based on regular expressions [20]. Drain applies a fixed-depth tree structure to represent log messages and extracts common templates effectively [7]. We choose Drain because of its accuracy, efficiency, and robustness (Figure 3).

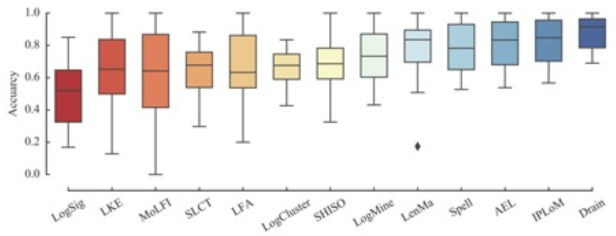


Fig. 3: Accuracy statistics of Log Parsers (from [7])

Tagtog is a collaborative text annotation platform to find, create, and maintain NLP datasets efficiently [21]. Tagtog also provides automatic text annotation using pre-selections, dictionaries, and machine learning models. Tagtog allows us to define our custom tags for entity extraction. When the dataset size and number of unique entities are large, using online annotation tools such as Tagtog is labor-intensive and time-consuming.

C. BERT for Named Entity Recognition Task

BERT (Bidirectional Encoder Representations from Transformers) is a Transformer-based machine learning technique for NLP pre-training developed by Google. BERT is designed to pre-train deep bidirectional representations from the unlabeled text by jointly conditioning the left and right context in all layers [22]. It scores new state-of-the-art results on eleven natural language processing tasks when it is introduced. The BERT framework has two stages: pre-training and fine-tuning. During pre-training, the model is trained on unlabeled data over special pre-training tasks such as Masked Language Modeling (MLM) [23], and Next Sentence Prediction (NSP) [24], [25]. Then the BERT model is initialized with the pre-trained parameters, and all parameters are fine-tuned using tagged data from the downstream tasks. In our case, this task is the NER task.

In this work, we use the bert-base-uncased model that is pre-trained on the lower-cased English text using a MLM objective. BERT base model has 12-encoder layers (Transformer blocks [26]), 768-hidden units, 12-attention-heads, 110M parameters.

Table I shows the BERT fine-tuning hyperparameters used in this paper.

Hyperparameter	Value
batch size	8
epochs	2
learning rate	3e-5
weight decay rate	0.01, 0.0
optimizer	AdamW

TABLE I: Hyperparameters for BERT fine-tuning

D. ELK Stack

Logstash pipeline is created to transform the data and send it to Elasticsearch. Kibana is used to build a dashboard and explore the datasets. Visualizations on the Kibana dashboard help to get information about the most common events, servers, hostnames, and IP addresses. We are planning to create a real-time streaming dashboard using Kibana for log analysis.

IV. Experimental Setup

We are working on Mac and Linux Operating Systems log files obtained from the Loghub repository, which is freely accessible for research. MacOS log file contains 102,786 log lines for a week from 1st July 09:00:55 to 8th July 08:52:05, while Linux OS (Operating System) log file contains 25,567 entities from 9th June 06:06:20 to 28th Feb to 04:48:54 (Table II).

	Mac	Linux
The number of log lines	102786	25567
Size of the log file	16 MB	2.24 MB

TABLE II: The Size of Datasets

As we mentioned before, there is no tagged log dataset on the internet to train our model, and manually tagging logs is time-consuming and labor-intense. Therefore, we used Drain online log parsing tool to parse and get a structured dataset in CSV format and then use column names as the labels. The Drain requires a log file format to parse it based on that format. MacOS log file follows-up the following form:

$$< Month >< Date >< Time >< User >< Component >$$
$$[< PID >] (< Address >): < Content >$$

Figure 4 shows MacOS log lines before and after applying the Drain log parsing tool. Then we add a "raw" named column, which consists of unstructured raw log lines to use parsed entities as the labels. In the pre-processing step, some columns like "Line ID" is dropped, time entries (month, date, and time) are merged, log entries such as timestamp, user, component, address, and content (log message) are extracted from Drain are defined as the labels, while a raw attribute is defined as the log.

The sample of the log:

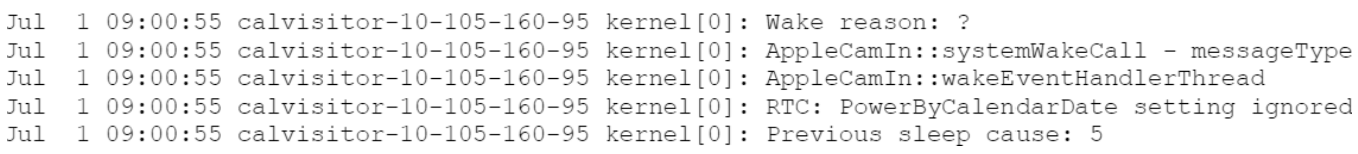
```
Jul 1 09:00:55 calvisitor-10-105-160-95
kernel[0]: AppleThunderboltGenericHAL::earlyWake
-complete - took 0 milliseconds
```

The corresponding tags for the above log:

```
['Timestamp', 'Timestamp', 'Timestamp', 'User',
   'other', 'Content', 'Content', 'Content',
  'Content', 'Content', 'Content', 'Content',
'Content', 'Content']
```

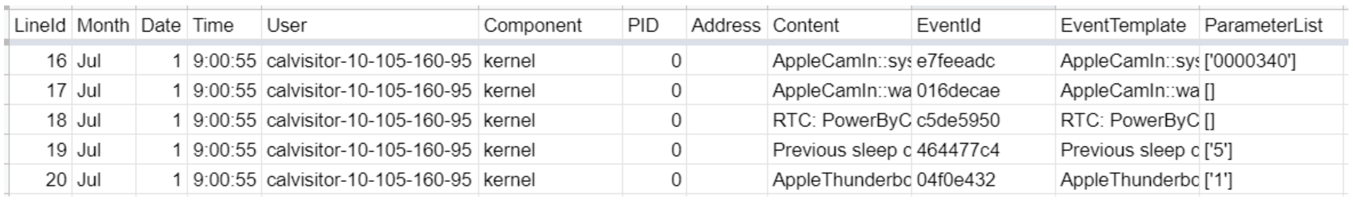
Before the fine-tuning, we need to prepare the dataset for PyTorch and the BERT. The log lines are tokenized with a bert-base-uncased pre-trained tokenizer. BERT tokenizer splits rare word, words with prefix and suffix into the several pieces, and sign them with the "##" symbol. However, we want to get exact labels for the log lines, and we define the function that repeats the actual label of the number of sub-words, instead of tagging sub-words with "other" or "X".

After that, string tags are encoded, and the attention mask is used to ignore the paddings. Dataset is split into



```
Jul  1 09:00:55 calvisitor-10-105-160-95 kernel[0]: Wake reason: ?
Jul  1 09:00:55 calvisitor-10-105-160-95 kernel[0]: AppleCamIn::systemWakeCall - messageType
Jul  1 09:00:55 calvisitor-10-105-160-95 kernel[0]: AppleCamIn::wakeEventHandlerThread
Jul  1 09:00:55 calvisitor-10-105-160-95 kernel[0]: RTC: PowerByCalendarDate setting ignored
Jul  1 09:00:55 calvisitor-10-105-160-95 kernel[0]: Previous sleep cause: 5
```

(a) Before the log parsing



LineId	Month	Date	Time	User	Component	PID	Address	Content	EventId	EventTemplate	ParameterList
16	Jul	1	9:00:55	calvisitor-10-105-160-95	kernel	0		AppleCamIn::sys	e7feeadc	AppleCamIn::sys	['0000340']
17	Jul	1	9:00:55	calvisitor-10-105-160-95	kernel	0		AppleCamIn::wa	016decae	AppleCamIn::wa	[]
18	Jul	1	9:00:55	calvisitor-10-105-160-95	kernel	0		RTC: PowerByC	c5de5950	RTC: PowerByC	[]
19	Jul	1	9:00:55	calvisitor-10-105-160-95	kernel	0		Previous sleep c	464477c4	Previous sleep c	['5']
20	Jul	1	9:00:55	calvisitor-10-105-160-95	kernel	0		AppleThunderbc	04f0e432	AppleThunderbc	['1']

(b) After the log parsing

Fig. 4: MacOS log file before and after the parsing

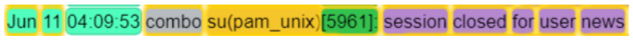
training and validation sets, where encoded labels are inputs, and 10% of the dataset is used for validation.

BertForTokenClassification class and AdamW optimizer is used to set up the BERT model for the fine-tuning (Table I).

The next step is fitting the model for NER. The paper [22] suggests using 2-4 epochs to reduce the learning rate linearly, and we use two epochs in the neural networks. We evaluate and plot the performance on the validation set.

Finally, we apply the model to extract the named entities from the new raw log lines.

We do the same procedures for the Linux OS log file to implement BERT for NER. Preprocessing is different as we are using the Tagtog annotation tool to tag the log file (Figure 5).



```
Jun 11 04:09:53 combo su(pam_unix)[5961]: session closed for user news
```

Fig. 5: An example of log labeling with Tagtog

After some preprocessing steps, we get the dataset with the columns of ”line Id”, ”log”, and ”tag”. The class is created to get the log lines based on line id; and take log entities and their corresponding labels from these loglines.

V. Results and Discussion

In this section, we discuss the results of the experimental setup.

A. Evaluation Metrics

The standard metrics such as Precision, Recall, F1-Score, and Accuracy Score are used. Precision shows the percentage of true labels among all the classifications into that class; Recall measures the percentage of true labels in the dataset that actually belongs to that label; F1-Score is the harmonic mean of the precision and recall.

$$Precision = \frac{TP}{TP + FP} \qquad Recall = \frac{TP}{TP + FN}$$

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$F1 = \frac{2 * precision * recall}{precision + recall}$$

B. Experimental Results

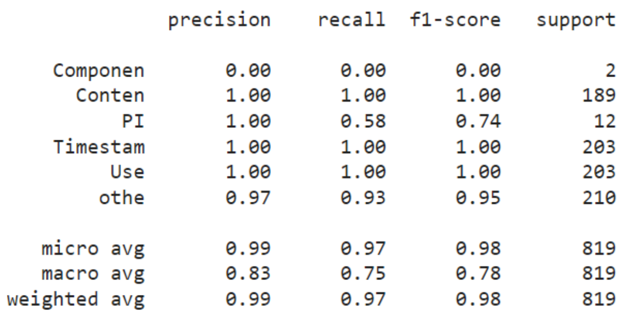
The datasets are split into the train and validation sets, and 90% of them are used for training, while the remaining 10% are used for the validation. BERT model achieves a 99% accuracy score and 98% F1-score on the MacOS log file. The validation accuracy is 99%, validation loss is 1%, and the accuracy score is 98% for the BERT model on the Linux OS dataset. The classification report is described in Figure 6, and the learning curves are shown in Figure 7.

Then we randomly pick the logline from the Linux OS dataset and want our model to identify named entities (Table III):

```
    Jun 11 04:03:19 combo su(pam_unix)[4718]:
session closed for user cyrus
```

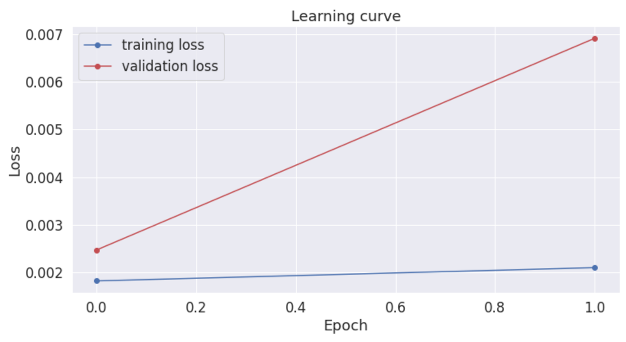
Then the result of each log entry is formatted as follows (Figure 8).

Generally, the BERT model achieves good results even with small datasets. Log files are records of a particular system or application. Therefore, they follow

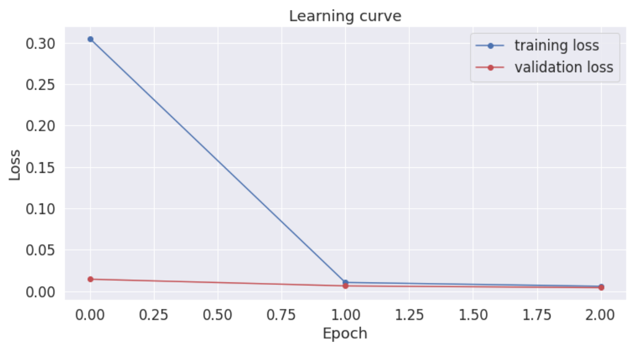


```
              precision    recall  f1-score   support

    Componen       0.00      0.00      0.00         2
      Conten       1.00      1.00      1.00       189
          PI       1.00      0.58      0.74        12
    Timestam       1.00      1.00      1.00       203
         Use       1.00      1.00      1.00       203
        othe       0.97      0.93      0.95       210

   micro avg       0.99      0.97      0.98       819
   macro avg       0.83      0.75      0.78       819
weighted avg       0.99      0.97      0.98       819
```

Fig. 6: Classification report of BERT on MacOS log file



(a) The Learning Curve of BERT model on MacOS log file

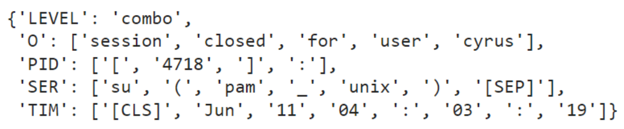


(b) The Learning Curve of BERT model on Linux OS log file

Fig. 7: Learning curves

Label	Token
TIM	[CLS]
TIM	Jun
TIM	11
TIM	04
TIM	:
TIM	03
TIM	:
TIM	19
LEVEL	combo
SER	su
SER	(
SER	pam
SER	_
SER	unix
SER	)
PID	[
PID	4718
PID	]
PID	:
O	session
O	closed
O	for
O	user
O	cyrus
SER	[SEP]

TABLE III: The named entities of the log line



```
{'LEVEL': 'combo',
 'O': ['session', 'closed', 'for', 'user', 'cyrus'],
 'PID': ['[', '4718', ']', ':'],
 'SER': ['su', '(', 'pam', '_', 'unix', ')', '[SEP]'],
 'TIM': ['[CLS]', 'Jun', '11', '04', ':', '03', ':', '19']}
```

Fig. 8: The result of the BERT model

a specific log format, and often entities such as the same component, hostname, timestamp fields are repeated in a log file. It is clear from Table III that we successfully extract named entities with the BERT model.

VI. Conclusion

Log files are valuable sources for real-time analysis and security thread detection. However, as a semi-structured data source, it first needs to be converted into a structured format. Although this can be done using regex expressions, they are highly domain specific and not easy to adapt other systems. We approach this problem by employing deep learning models for Named Entity Recognition (NER). Our results show that we can successfully extract entities from macOS, and Linux OS log files using the BERT model with 99% accuracy. Furthermore, using machine learning algorithms allow us to adapt our solution to different systems by transferring or re-training the models.

Future work includes training BERT model on more datasets and building an annotation tool for real-time streaming log entries.

References

[1] I. O. de Urbina Cazenave, E. Köşlük, and M. C. Ganiz, “An anomaly detection framework for bgp,” in *2011 International Symposium on Innovations in Intelligent Systems and Applications*, pp. 107–111, IEEE, 2011.

[2] A. Brown, A. Tuor, B. Hutchinson, and N. Nichols, “Recurrent neural network attention mechanisms

for interpretable system log anomaly detection," in *Proceedings of the First Workshop on Machine Learning for Computing Systems*, pp. 1–8, 2018.

[3] A. D. Kent, "Cyber security data sources for dynamic network research," in *Dynamic Networks and Cyber-Security*, pp. 37–65, World Scientific, 2016.

[4] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1285–1298, 2017.

[5] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[6] S. He, J. Zhu, P. He, and M. R. Lyu, "Loghub: a large collection of system log datasets towards automated log analytics," *arXiv preprint arXiv:2008.06448*, 2020.

[7] J. Zhu, S. He, J. Liu, P. He, Q. Xie, Z. Zheng, and M. R. Lyu, "Tools and benchmarks for automated log parsing," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pp. 121–130, IEEE, 2019.

[8] H. Studiawan, F. Sohel, and C. Payne, "Automatic log parser to support forensic analysis," 2018.

[9] P. Pokharel, *Information Extraction Using Named Entity Recognition from Log Messages*. PhD thesis, Pokhara University, 2018.

[10] J. Li, A. Sun, J. Han, and C. Li, "A survey on deep learning for named entity recognition," *IEEE Transactions on Knowledge and Data Engineering*, 2020.

[11] D. Nadeau and S. Sekine, "A survey of named entity recognition and classification," *Lingvisticae Investigationes*, vol. 30, no. 1, pp. 3–26, 2007.

[12] O. Etzioni, M. Cafarella, D. Downey, A.-M. Popescu, T. Shaked, S. Soderland, D. S. Weld, and A. Yates, "Unsupervised named-entity extraction from the web: An experimental study," *Artificial intelligence*, vol. 165, no. 1, pp. 91–134, 2005.

[13] S. Sekine and C. Nobata, "Definition, dictionaries and tagger for extended named entity hierarchy.," in *LREC*, Lisbon, Portugal, 2004.

[14] S. Zhang and N. Elhadad, "Unsupervised biomedical named entity recognition: Experiments with clinical and biological texts," *Journal of biomedical informatics*, vol. 46, no. 6, pp. 1088–1098, 2013.

[15] A. Baevski, S. Edunov, Y. Liu, L. Zettlemoyer, and M. Auli, "Cloze-driven pretraining of self-attention networks," *arXiv preprint arXiv:1903.07785*, 2019.

[16] L. Cui and Y. Zhang, "Hierarchically-refined label attention network for sequence labeling," *arXiv preprint arXiv:1908.08676*, 2019.

[17] E. F. Tjong Kim Sang and F. De Meulder, "Introduction to the conll-2003 shared task: Language-independent named entity recognition," in *Proceedings of CoNLL-2003* (W. Daelemans and M. Osborne, eds.), pp. 142–147, Edmonton, Canada, 2003.

[18] L. Derczynski, E. Nichols, M. van Erp, and N. Limsopatham, "Results of the wnut2017 shared task on novel and emerging entity recognition," in *Proceedings of the 3rd Workshop on Noisy User-generated Text*, pp. 140–147, 2017.

[19] J. Bos, V. Basile, K. Evang, N. J. Venhuizen, and J. Bjerva, "The groningen meaning bank," in *Handbook of linguistic annotation*, pp. 463–496, Springer, 2017.

[20] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Drain: An online log parsing approach with fixed depth tree," in *2017 IEEE International Conference on Web Services (ICWS)*, pp. 33–40, IEEE, 2017.

[21] J. M. Cejuela, P. McQuilton, L. Ponting, S. J. Marygold, R. Stefancsik, G. H. Millburn, B. Rost, F. Consortium, *et al.*, "tagtog: interactive and text-mining-assisted annotation of gene mentions in plos full-text articles," *Database*, vol. 2014, 2014.

[22] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[23] W. L. Taylor, ""cloze procedure": A new tool for measuring readability," *Journalism quarterly*, vol. 30, no. 4, pp. 415–433, 1953.

[24] Y. Jernite, S. R. Bowman, and D. Sontag, "Discourse-based objectives for fast unsupervised sentence representation learning," *arXiv preprint arXiv:1705.00557*, 2017.

[25] L. Logeswaran and H. Lee, "An efficient framework for learning sentence representations," *arXiv preprint arXiv:1803.02893*, 2018.

[26] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *arXiv preprint arXiv:1706.03762*, 2017.