# Automatic log parser to support forensic analysis

Hudan Studiawan

Ferdous Sohel

Christian Payne

DOI: 10.25958/5c5268c766686

# AUTOMATIC LOG PARSER TO SUPPORT FORENSIC ANALYSIS

Hudan Studiawan, Ferdous Sohel, Christian Payne
School of Engineering and Information Technology, Murdoch University, Perth, Australia
{hudan.studiawan, f.sohel, c.payne}@murdoch.edu.au

## Abstract

*Event log parsing is a process to split and label each field in a log entry. Existing approaches commonly use regular expressions or parsing rules to extract the fields. However, such techniques are time-consuming as a forensic investigator needs to define a new rule for each log file type. In this paper, we present a tool, namely nerlogparser, to parse the log entries automatically, where log parsing is modeled as a named entity recognition problem. We use a deep machine learning technique, specifically the bidirectional long short-term memory networks, as the underlying architecture for this purpose. Unlike existing tools, nerlogparser is a fully automatic tool as the investigators do not need to define any parsing rules and it is generic as there is only one model to parse various types of log files. Experimental results show that nerlogparser achieves superior performance compared with other traditional machine learning methods.*

**Keywords**
log parser, log forensics, named entity recognition, long short-term memory

## INTRODUCTION

In a forensic analysis, the first step before investigating event logs is to parse a log file. Event log parsing is a mechanism to separate each field in a log entry. Afterward, a label is given to each split field. Event log parsing is needed for general log analysis as the forensic investigator requires data from log entries such as timestamp, hostname, service name, or IP address. In addition, the main message is also important as it contains further information such as errors, alerts, or some informational entries. An illustration of event log parsing is shown in Figure 1.

Existing solutions for log parsing include applying regular expressions (regex) and PyParsing (McGuire, 2007). For example, regex is used to extract each fields in a syslog before running forensic event correlation (Chen et al., 2003). Schatz et al. (2004) uses regex for handling the unstructured and heterogeneous logs before they are fed into a forensic event knowledge base. In an event reconstruction case, the investigator also need to define a pattern in log files. In Gunestas et al. (2016), regex is used to identify events from a message in a log entry. After acquiring a well-structured log, the extracted events can be examined to check for a misuse activity or attack.

Furthermore, the investigator has to define regular expression rules in the first place to get each log entry chunk before running the analyses. Such analyses include event log abstraction or template generation and anomaly detection (He et al., 2016). Another parsing rule is PyParsing that has more human-readable rules than regex. For example, syntax `Word(alphas)` will parse a string containing all lowercase and uppercase letters. In Lemoudden et al. (2016), the use of PyParsing is demonstrated to extract the fields from a Hadoop log and then save them in the JavaScript Object Notation (JSON) format.

The problem with existing approaches is that the forensic investigator needs to define regex or PyParsing rules for every log file. In some cases, the same type of log files can have different regex as the system administrator can set different configurations to the file. It is very time-consuming and there is a need to maintain a large list of regex or PyParsing rules to parse log files.

In terms of log format, we deal with semi-structured log files in this paper. We define a *semi-structured* format as log file that has a structure but not one that is explicitly stated as it is only separated by space or tab characters. Another type of log format is the *structured* format that has an exact template. The example of a structured log format is Windows logs as the entries are saved in an XML file. The parser for the structured format is usually available from the vendor. For instance, the parser of Windows event logs is LogParser as an official log parser from Microsoft (Microsoft, 2005) or python-evtx (Ballenthin, 2018) that developed by the open source community. The proposed tool in this paper aims to automatically split and label each field in a semi-structured log file.

We model log parsing as named entity recognition problem (Tjong Kim Sang & De Meulder, 2003). This issue is a popular topic in natural language processing research area and involves identifying an entity in a text such as a person, organization, and country name. We use a deep machine learning technique to train various log files such

```
Jan 18 09:33:03 victoria init: Switching to runlevel: 0
─────────────── ──────── ───── ────────────────────────
   timestamp    hostname service        message
```

*Figure 1. An example of event log parsing from a syslog file (Arcas et al., 2011)*

as Linux logs, web server logs, and honeypot logs. We chose the bidirectional long short-term memory (BLSTM) approach (Graves & Schmidhuber, 2005) because this architecture is originally designed to detect sequences. In the case of a log entry, the sequence refers to the fields such as timestamp, hostname, service name, and the main message. The bidirectional model is used to increase the detection performance as the procedure runs in both forward and backward context in the sequence.

In addition, an embedding technique is employed at the beginning of the BLSTM layer to give a better word and character-level representation of the log inputs before processed. Finally, we propose a tool called **nerlogparser** that stands for **n**amed **e**ntity **r**ecognition for event **log parser**.

To summarize, the main contributions of this paper are:

1. We propose nerlogparser, an automatic tool for event log parsing that:
    a. does not require forensic investigators to define any parsing rules; and
    b. provides genericity as there is only one model file to parse various types of log files;
2. The application of deep learning techniques, specifically BLSTM, to parse event logs provide the best performance of the techniques tested with 99.98% accuracy.

## TERMINOLOGY

In this section, we define terminology used in this paper. An *event* is an identifiable action that occurs on a device such as a computer and it is recorded in a log entry (European Commission, 2010). An *event log* is a record of events, usually represented in a log file. A *log file* is a file that records activities from applications or operating system. A *log entry* is a single record in a log file. An *event message* refers to the main message in a log entry excluding timestamp and any other fields.

A *field* or an *entity* is a meaningful part in a log entry and can contain one or more words such as IP address and service name. A *label* or *entity name* is the name for a particular field or entity. Note that event log parsing discussed in this paper is different from event log abstraction or template generation as described in He et al. (2016). We define *log parsing* as a process to split and label each field in a log entry. Meanwhile, log parsing in He et al. (2016) generates abstractions or templates for entries from a given log file.

## THE PROPOSED EVENT LOG PARSER: nerlogparser

Parsing is the first step before analyzing log-related forensic data as shown in Figure 2. To assist a forensic investigator to parse log files, we propose an automatic event log parser to make log easier to examine. First, we model log parsing as named entity recognition problem. Second, we explain the input representation of log entries. Finally, we depict a deep learning architecture to automatically parse log files. Each step is described in detail in the following sections.

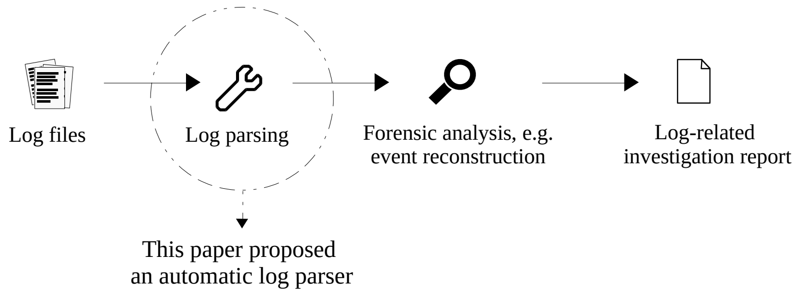

Log files
Log parsing
Forensic analysis, e.g. event reconstruction
Log-related investigation report
This paper proposed an automatic log parser

*Figure 2. Log parsing position in a typical forensic analysis*

## Log Parsing as Named Entity Recognition Problem

The original definition of named entities are phrases that contain the names of persons, organizations, and locations (Tjong Kim Sang & De Meulder, 2003). Extracting named entities from a text is commonly called as named entity recognition (NER). In the context of log files, we redefine named entities as a word or phrases containing common fields such as timestamp, hostname, service name, or IP address. In a log parser case, recognizing named entities is equivalent with identifying each field in a log entry.

The most common format of NER is IOB (Inside-Outside-Begin) tagging (Tjong Kim Sang & De Meulder, 2003). The `I-XXX` tag is for words inside a named entity of type `XXX`. If there are two words of type `XXX` are located next to each other, the first word is tagged with `B-XXX` to confirm that it is a start of an entity. Furthermore, `O` tag is for words that outside of predefined named entities. An example of named entities in a log entry from Figure 1 with IOB annotation is shown in Figure 3.

From Figure 3, we can easily decode the notation that `I-TIM` means timestamp, while `B-TIM` is the first word in the entity, `I-HOS` means hostname, and `I-SER` is the service name. Each word in the main message is tagged as `O` because it is outside of defined entities.

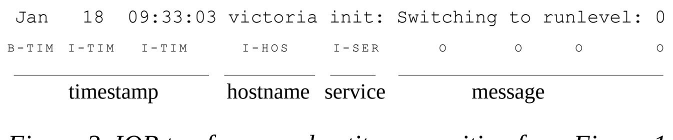
NER is a classification problem and existing NER methods use traditional machine learning approaches such as naïve Bayes (Banko et al., 2007), perceptron (Carreras et al., 2003), stochastic gradient descent (Tsuruoka, 2009), and passive-aggressive classifier (Cherry & Guo, 2015). Features are extracted from the data instead of the raw data are fed to the classifier. In machine learning, a feature is a measurable property of the observed data. The common predefined features used in traditional machine learning for NER are a particular word $w_i$ as an entity itself, part of speech (POS) of word $POS_i$, $w_{i+1}$ as the next word, $POS_{i+1}$, $w_{i+2}$, $POS_{i+2}$, $w_{i-1}$ as the previous word, $POS_{i-1}$, $w_{i-2}$, and $POS_{i-2}$. POS is a category to define how a particular word is used in a sentence, such as noun and verb. These features suggest that a specific entity will appear in a special position in a sentence indicated by its surrounding words.

On the other hand, deep learning has emerged as a new approach in NER research area (Chiu & Nichols, 2016; Lample et al., 2016; Rei, 2017). The main difference between traditional machine learning approaches and deep learning is that we do not need to define the features of the input data. Deep learning techniques can learn features in an incremental fashion. Another advantage of deep learning is that it provides an end-to-end solution. In the learning or training phase, end-to-end means that the developer will only need to give the input data and basic initial parameters for training. While in the testing phase, the users only supply the data and they will be parsed based on the models generated in the learning phase. We use a deep learning technique for NER for log parsing as explained in the next sections.

## Word and Character Embedding as Input Representation

Raw data needs to be prepared so that it can be fed into a deep learning network as an input. The text log entries should be converted into vectors of numbers. Therefore, we use word embedding to represent raw log entries as vectors. Word embedding can accommodate meaning, a relationship between words, and context in a text (Pennington et al., 2014). Every word will be represented as a unique embedding. In the training phase, nerlogparser starts with taking the input of one sentence, split the sentence by space, and it is called tokens. The log entries that have been processed into tokens are converted to a sequence of word embedding.

We use GloVe (Pennington et al., 2014) as pretrained word embedding. Specifically, we used version glove.6B 300 dimensions, which is trained on 6 billion tokens. It is trained on Wikipedia English data (Wikimedia, 2014) and English Gigaword (Parker et al., 2011) containing news data corpus collected over several years. GloVe trains the corpus and gets the embedding of each word based on statistical properties such as word occurrence in a context or a sentence. GloVe is employed to initialize the look up dictionary for word embedding. We chose GloVe as word embedding because it uses Wikipedia data and news corpus, which are representative enough to get the similarity and context between words. Vocabulary used in the training of GloVe are common words and it uses six billion words in total, so we do not have any problem when using it in the log data.



```
Jan    18  09:33:03 victoria init: Switching to runlevel: 0
B-TIM I-TIM  I-TIM     I-HOS    I-SER     O      O     O      O
```

*Figure 3. IOB tag for named entity recognition from Figure 1*

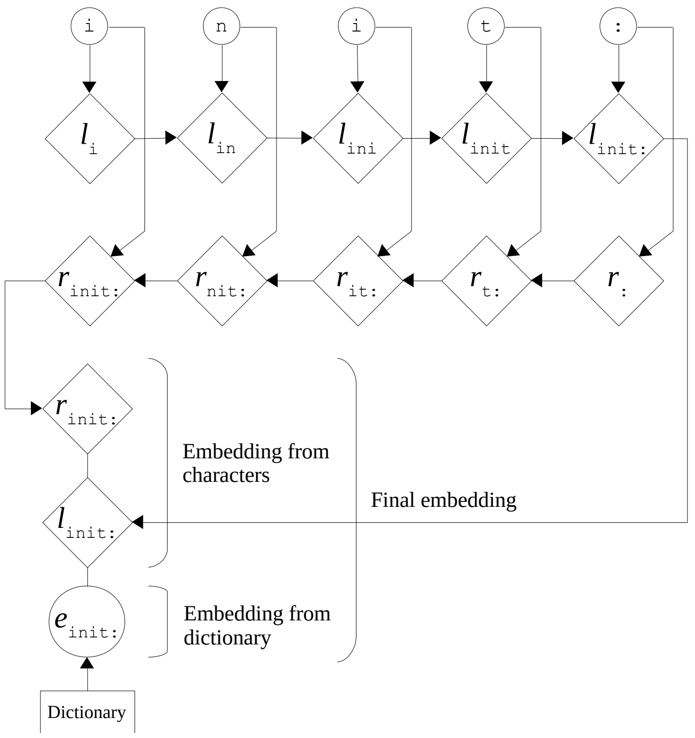


*Figure 4. Character and word embedding for word* `init:` *based on a model by Lample et al. (2016)*

We investigate another input representation namely character embedding. It is similar to word embedding but the representation is at a character-level. The advantage of character embedding is that it learns the prefix and suffix for a particular word. Therefore, it will provide a more representative and adaptable model for a given log entry format. Illustration for word and character embedding as data input representation is depicted in Figure 4 where *l* is a left context, *r* is a right context, and *e* is an embedding.

We use the character embedding model proposed in (Lample et al., 2016). A dictionary containing characters and its embedding is constructed. The embedding for each character in a particular word are set in both direction order for a forward and a backward LSTM, respectively. Moreover, a word embedding is generated from its characters by concating its forward and backward model of BLSTM. The character-based model is then concatenated with a word-level model from a word lookup dictionary. In the testing phase, any words that do not have an embedding are set to a UNK (unknown) embedding. We set the hidden dimension by 100 for the forward and backward character LSTM, respectively. Therefore, the character-based model of words has a dimension of 200.

## Bidirectional Long Short-Term Memory (BLSTM) as the Main Architecture

We assign an entity name to every token in a log entry using a bidirectional LSTM (BLSTM) (Graves & Schmidhuber, 2005). BLSTM is derived from recurrent neural networks (RNN) (Goller & Kuchler, 1996), which is a variant of neural networks designed to process sequential data. In event log parsing, the sequences are each field in a log entry such as timestamp, hostname, service name, and IP address.

The input of RNN is a sequence of vectors and it yields another sequence of information or label about the input sequence. RNN can learn long dependencies between sequences but it tends to have a bias with recent input sequences and also has gradient vanishing problems (Bengio et al., 1994). LSTM (Hochreiter & Schmidhuber, 1997) solves this problem by providing a memory-cell to accommodate long dependencies. An LSTM memory-cell unit is composed of several gates, which control the proportions of information to forget and to pass on to the next step. We follow these standard equations from (Hochreiter & Schmidhuber, 1997) to update an LSTM unit at time $t$:

$$
\begin{aligned}
\mathbf{i}_t &= \sigma(\mathbf{W}_i\mathbf{h}_{t-1} + \mathbf{U}_i\mathbf{x}_t + \mathbf{b}_i) \\
\mathbf{f}_t &= \sigma(\mathbf{W}_f\mathbf{h}_{t-1} + \mathbf{U}_f\mathbf{x}_t + \mathbf{b}_f) \\
\tilde{\mathbf{c}}_t &= \tanh(\mathbf{W}_c\mathbf{h}_{t-1} + \mathbf{U}_c\mathbf{x}_t + \mathbf{b}_c) \\
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t \\
\mathbf{o}_t &= \sigma(\mathbf{W}_o\mathbf{h}_{t-1} + \mathbf{U}_o\mathbf{x}_t + \mathbf{b}_o) \\
\mathbf{h}_t &= \mathbf{o}_t \odot \tanh(\mathbf{c}_t)
\end{aligned}
$$

where $\mathbf{h}_t$ is the hidden state vector storing information at time $t$ and $\mathbf{x}_t$ is the word embedding as the input vector. The operator $\sigma$ is the element-wise sigmoid function and $\odot$ is the element-wise product. $\mathbf{W}_i, \mathbf{W}_f, \mathbf{W}_c, \mathbf{W}_o$ denote the weight matrices for hidden state $\mathbf{h}_t$, while $\mathbf{U}_i, \mathbf{U}_f, \mathbf{U}_c, \mathbf{U}_o$ are weight matrices for different gates for $\mathbf{x}_t$. Lastly, $\mathbf{b}_i, \mathbf{b}_f, \mathbf{b}_c, \mathbf{b}_o$ are the bias vectors.

From the LSTM equation, there are two inputs of each direction of LSTM: 1) the hidden state from the previous time step $\mathbf{h}_{t-1}$ and 2) the embedding from the current step $\mathbf{x}_t$. Afterwards, it produces a new hidden state. An original LSTM $\overrightarrow{\mathbf{h}_t}$ moves from left to right or forward direction to learn the sequences. To get the reverse context, one can add another LSTM $\overleftarrow{\mathbf{h}_t}$ that processes sequences from right to left or backward direction. These two directions will provide much more information about sequence and it is called bidirectional LSTM (Graves & Schmidhuber, 2005). The hidden representations from both LSTMs are concatenated to get a context for each word on a log entry in both directions, $\mathbf{h}_t = [\overrightarrow{\mathbf{h}_t}, \overleftarrow{\mathbf{h}_t}]$. Illustration of BLSTM architecture for a log entry without the main message is shown in Figure 5. Each word becomes an element a sequence vector and the output is each respective label such as `Jan: B-TIM, 18: I-TIM, 09:33:03: I-TIM` where the label follows IOB tagging format.

In NER problem, there are strong dependencies between adjacent labels and it should be beneficial to take advantage of these connections. Thus, BLSTM gives an effective representation of each field that catches a context in the surrounding words in a log entry. Unlike traditional machine learning that needs features definition before training, BLSTM learns features based on both context directions. To predict an entity label for each token, nerlogparser uses a softmax output layer. In the softmax layer, BLSTM predicts a normalized distribution over all possible labels for every token.

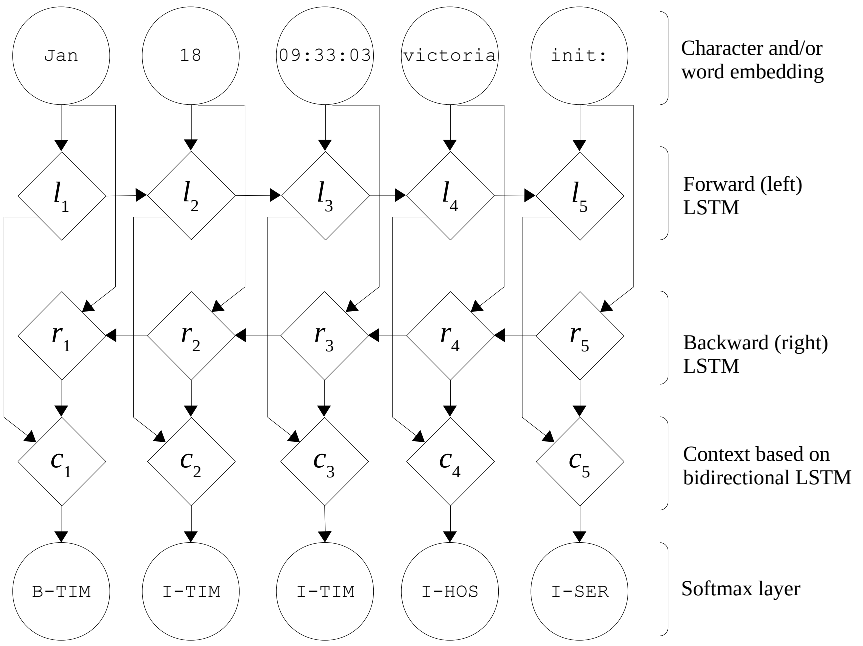

Jan
18
09:33:03
victoria
init:
Character and/or word embedding
$l_1$ $l_2$ $l_3$ $l_4$ $l_5$
Forward (left) LSTM
$r_1$ $r_2$ $r_3$ $r_4$ $r_5$
Backward (right) LSTM
$c_1$ $c_2$ $c_3$ $c_4$ $c_5$
Context based on bidirectional LSTM
B-TIM
I-TIM
I-TIM
I-HOS
I-SER
Softmax layer

*Figure 5. An illustration of named entity recognition with BLSTM for a log entry excluding the main message*

## EXPERIMENTAL RESULTS

### Datasets

To test the performance and to get the generic model for nerlogparser, we use various types of log files from different environments such as the Linux operating system, supercomputer, distributed systems, and honeypots. We also use common logs such as web server logs. The different kinds of attacks recorded in the log files for training are not important. In log parsing, we look for a *various field structure* of log entries in many log files, rather than the content of the log message. Therefore, certain kinds of attacks or any undiscovered attacks do not limit the ability to train and test the log parser model. Accordingly, nerlogparser is not affected by the content of the log message.

The first dataset is log files from the BlueGene/L supercomputer system. This dataset was collected from Lawrence Livermore National Labs (LLNL) and contains alert and non-alert messages (Oliner & Stearley, 2007). The second dataset is the forensic image data from Digital Corpora namely *nps-2009-casper-rw* (Garfinkel, 2009). This image is an ext3 file system dump from a bootable USB. Therefore, we extract log files from the directory `/var/log/` to be analyzed.

Other datasets are coming from DFRWS (Digital Forensic Research Workshop) 2009 and 2016. In the DFRWS Forensic Challenge 2009 (Casey & Richard III, 2009), forensic researchers investigated Linux logs including authentication logs to trace the attacker in two hosts namely “jhuisi” and “nssal”. The more recent challenge (CMAND, 2016) provided a case in a Software-Defined Network (SDN) environment. The forensic investigators needed to carve and parse the log files from an OpenVSwitch (OVS) switch before examining the SDN event logs.

The Honeynet Project also offers some challenges that related to event log analysis. The Forensic Challenge 5 2010 (Marty et al., 2010) concerns a compromised Linux operating system. The directory `/var/log/` has been imaged and the investigator needs to analyze the brute-force attack recorded in the log files. The Forensic Challenge 7 2011 (Arcas et al., 2011) provides a disk image of a compromised Linux server and a forensic analysis is required in order to understand what really happened. From the provided hard disk dump, we extract the log files and parse them.

Furthermore, we also include log files from Kippo which is an SSH (secure shell) honeypot. A Kippo was installed and configured on a DigitalOcean droplet with public IP address so everyone including real attackers and botnets can reach this server. We have used this dataset previously in Studiawan et al. (2017). We also incorporate logs from Proxifier as a proxy client and Zookeeper as a distributed system coordinator. Both Proxifier and Zookeeper logs have been used in (He et al., 2016) to analyze event log abstraction.

In addition, we add web access logs from Security Repository (SecRepo) (Sconzo, 2018) as web logs are commonly analyzed by system administrators. SecRepo is a website that provides a list of the various security-related datasets including event logs. Web server logs from SecRepo are generated by the website visitors. We only take the first 10,000 lines in BlueGene/L and Zookeeper as these datasets are huge with total 4,747,963 and 74,380, respectively, for only one type of log file. The summary of datasets and their number of lines is displayed in Table 1.

*Table 1. List of Event Log Datasets*

| Dataset | # of lines |
|---|---:|
| BlueGene/L (Oliner & Stearley, 2007) | 10,000 |
| Digital Corpora casper-rw (Garfinkel, 2009) | 11,086 |
| DFRWS 2009 jhuisi host (Casey & Richard III, 2009) | 11,737 |
| DFRWS 2009 nssal host (Casey & Richard III, 2009) | 107,093 |
| DFRWS 2016 (CMAND, 2016) | 3,304 |
| Honeynet Challenge 5 (Marty et al., 2010) | 124,386 |
| Honeynet Challenge 7 (Arcas et al., 2011) | 8,712 |
| Kippo honeypot (Studiawan et al., 2017) | 10,094 |
| Proxifier (He et al., 2016) | 10,107 |
| SecRepo access logs (Sconzo, 2018) | 10,039 |
| Zookeeper (He et al., 2016) | 10,000 |

*Table 2. Training with 15 Epochs for nerlogparser*

| Epoch | Precision | Recall | F1 | Accuracy |
|---|---|---|---|---|
| 1 | 13.92 | 4.28 | 6.54 | 59.08 |
| 2 | 74.00 | 74.15 | 74.07 | 90.42 |
| 3 | 96.39 | 97.20 | 96.79 | 97.95 |
| 4 | 98.55 | 99.86 | 99.20 | 99.50 |
| 5 | 99.98 | 99.94 | 99.96 | 99.98 |
| 6 | 99.99 | 99.95 | 99.97 | 99.98 |
| 7 | 99.98 | 99.95 | 99.96 | 99.98 |
| 8 | 99.99 | 99.96 | 99.98 | 99.99 |
| 9 | 100.00 | 99.96 | 99.98 | 99.99 |
| 10 | 99.99 | 99.96 | 99.98 | 99.99 |
| 11 | 100.00 | 99.96 | 99.98 | 99.99 |
| 12 | 100.00 | 99.96 | 99.98 | 99.99 |
| **13** | **100.00** | **99.96** | **99.98** | **99.99** |
| 14 | 99.99 | 99.96 | 99.98 | 99.99 |
| 15 | 99.99 | 99.96 | 99.98 | 99.98 |

## Training of nerlogparser

For training and testing phase, nerlogparser is implemented in Python 3 and using TensorFlow (Abadi et al., 2016) as deep learning library. For traditional machine learning implementations that will be compared with nerlogparser, we use Natural Language Toolkit (NLTK) (Bird, 2009) and scikit-learn (Pedregosa et al., 2011). To train the all of the models, we use a computer with 12 cores of CPU, 64GB of RAM, and NVIDIA GeForce GTX 1080 8GB and GTX 1050 Ti 4GB GPUs. The output of training phase is a model file that can be used directly by a forensic investigator to parse log files.In the preprocessing step, each word is converted into lowercase and the digit character is converted to "NUM". As nerlogparser is a supervised learning method, we need to build labelled entries from the datasets. We use PyParsing (McGuire, 2007) to parse log entries and each field is labelled based on IOB tagging format. We split each file in the datasets in these proportions: 60% for training, 20% for development, and 20% for testing. These proportions are common practice in machine learning research and applications (Géron, 2017).

Hyperparameters used in the deep learning architectures are explained as follows. The maximum epoch for training is 15 with a dropout rate of 0.5. We use the batch size of 15 and early stopping for three times. This means that the training will stop if there is no more improvement after three epochs. We use Adam (Kingma & Ba, 2014) as the learning method in BLSTM architecture. Adam is good for general cases because it is efficient, only needs a small amount of RAM, and is invariant to gradient scaling. We set learning rate to 0.001 and learning rate decay to 0.9. For LSTM hidden states, we set it to 100 where character embedding is activated and 300 when only word embedding is defined.

Table 2 shows the training process to get the best evaluation metrics with 15 epochs. The best performance for nerlogparser model is achieved in 13th epoch with 100% precision, 99.96% recall, 99.98% F1, and 99.99% accuracy. As early stopping is set to three, the training stops when there is no improvement after three epochs or the total epochs is finished.

## Performance Evaluation

We test the performance of architecture combinations where BLSTM become the basis. We test BLSTM and char + BLSTM. The default input representation for BLSTM is a word embedding, while the default output is a softmax layer. "char + BLSTM" means that it uses both word and character embedding.

We compare the performance of nerlogparser with existing traditional machine learning methods for NER. The compared methods are multinomial naïve Bayes (Rennie et al., 2003), perceptron (Carreras et al., 2003), stochastic gradient descent (Tsuruoka et al., 2009), and passive-aggressive classifier (Cherry & Guo, 2015). However, the implementation is not exactly as stated in these papers as we use scikit-learn library and NLTK for this experiment. For traditional machine learning methods, we use the default parameter from scikit-learn. For non-default parameters, we set the batch size for log data to 500.

*Table 3. Comparison of Performance Metrics in Percent (%) for Traditional Methods and nerlogparser*

| Method | Precision | Recall | F1 | Accuracy |
|---|---|---|---|---|
| nerlogparser (BLSTM) | 99.82 | 99.65 | 99.73 | 99.82 |
| nerlogparser (char + BLSTM) | **99.98** | **99.94** | **99.96** | **99.98** |
| Perceptron | 75.80 | 89.30 | 82.00 | 93.30 |
| SGD | 91.00 | 65.00 | 75.80 | 80.90 |
| Multinomial NB | 93.70 | 94.70 | 94.20 | 96.90 |
| Passive-aggresive | 99.00 | 98.70 | 98.90 | 99.30 |

The evaluation metrics used to measure the performance of the methods are precision, recall, F1 score, and accuracy. The results of this comparison are depicted in Table 3 where the bold value indicates the best performance. Based on the results shown in Table 3, char + BLSTM gives the best performance with even values for all metrics: 99.98% precision, 99.94% recall, 99.96% F1 score, and 99.98% accuracy.

In our experiments, the traditional machine learning methods are inferior to the deep learning technique because they use handcrafted features. Perceptron gives fair results with precision of 75.80%, recall of 89.30%, and F1 scored 82.00%, with an accuracy of 93.30%. This is because the output from the first layer becomes the input to the second layer. Therefore, the classification errors in the first layer will influence the accuracy of the second.

SGD provides the worst performance compared to other methods with only 75.80% F1 score and 80.90% accuracy. SGD is sensitive to feature scaling, so it needs a very large dataset to give the best model. It seems that the dataset used in this experiment is not large enough for SGD. Even if the size of the datasets were increased, it is unlikely that SGD would outperform nerlogparser as the BLSTM-based method are known to also perform better when supplied with more data.

The improved version naïve Bayes called multinomial naïve Bayes also gives fair results with an F1 score of 94.20% and accuracy of 96.90%. The Bayesian-based method assumes that the feature is independent of each other. In NER case for log data, this assumption is not always true as each entry is highly related to surrounding entries. Besides, data input are assumed to be in normal distribution in naïve Bayes and power-law distribution in multinomial naïve Bayes. In real-world log files, the data distribution is not always same as assumed in those methods.

Passive-aggressive classifier provides excellent results and the best for traditional machine learning with 98.90% and 99.30% for F1 score and accuracy, respectively. The memory requirement of passive-aggressive is minimized and it can run in online mode meaning that it updates the model one sentence at a time. The passive-aggressive is a large-margin learning algorithm that attempts to separate correct and incorrect sequences. The performance metrics are comparable to the BLSTM-based method. However, nerlogparser can achieve the accuracy very close to 100% and this performance is critical to parse event logs to avoid parsing errors.

Compared to the aforementioned methods and BLSTM-only architecture, nerlogparser with "char + BLSTM" model provides the best performance. The reason is that the combination of character and word embedding gives a good vector representation for log entries as it learns the prefix, suffix, left and right adjacent token of a particular word. As BLSTM was designed to process sequences, it also supports the excellent performance of nerlogparser as we model a log entry as a sequence of fields that we assign labels to each of them.

## CONCLUSION AND FUTURE WORK

In this paper, we propose nerlogparser, an automatic event log parser based on the named entity recognition model. We use bidirectional long short-term memory to detect a sequence of fields in a log entry such as timestamp, hostname, IP address, service name, and the main message. To increase the accuracy, we also employ character-level embedding in the input layer of BLSTM.

The forensic investigators can use pretrained nerlogparser model to parse log files. Note that as the proposed tool is automatic, the end users do not need to provide any input parameters. In addition, investigators can train their own domain-specific log files. As presented nerlogparser is trained on log files from some common systems such as Linux, web servers, and SSH honeypot.

In future, we will run training on more various types of log files to make nerlogparser more generic. For instance, there are 16 datasets in Loghub repository (LogPAI, 2018) and we only use three of them. We also plan to train word embedding in event log data as existing pre-trained word embedding we use such as GloVe (Pennington et al., 2014) use Wikipedia data and news corpus to model the embedding.

## ACKNOWLEDGMENT

This work is supported by The Indonesia Lecturer Scholarship (BUDI) from Indonesia Endowment Fund for Education (LPDP), Republic of Indonesia.

## REFERENCES

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., & Zheng, X. (2016). TensorFlow: a system for large-scale machine learning. In *Proceedings of the 12th USENIX conference on Operating Systems Design and Implementation* (pp. 265-283).

Arcas, G., Gonzales, H., & Cheng, J. (2011). Challenge 7 of the Honeynet Project Forensic Challenge 2011. Retrieved from https://www.honeynet.org/challenges/2011_7_compromised_server

Ballenthin, W. (2018). python-evtx: Pure Python parser for recent Windows event log files. Retrieved from https://github.com/williballenthin/python-evtx

Banko, M., Cafarella, M. J., Soderland, S., Broadhead, M., & Etzioni, O. (2007). Open information extraction from the web. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence* (pp. 2670-2676).

Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, *5*(2), 157-166.

Bird, S., Klein, E., & Loper, E. (2009). *Natural language processing with Python: Analyzing text with the Natural Language Toolkit*. O'Reilly Media, Inc.

Carreras, X., Màrquez, L., & Padró, L. (2003). Learning a perceptron-based named entity chunker via online recognition feedback. In *Proceedings of the 7th Conference on Natural Language Learning at HLT-NAACL* (pp. 156-159).

Casey, E. & Richard III, G. G. (2009). DFRWS Forensic Challenge 2009. Retrieved from http://old.dfrws.org/2009/challenge/index.shtml

Chen, K., Clark, A., De Vel, O., Mohay, G. (2003). ECF - Event correlation for forensics. In *Proceedings of the 1st Australian Computer Network and Information Forensics Conference* (pp. 1–10).

Cherry, C., & Guo, H. (2015). The unreasonable effectiveness of word representations for twitter named entity recognition. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (pp. 735-745).

Chiu, J. P., & Nichols, E. (2016). Named Entity Recognition with Bidirectional LSTM-CNNs. *Transactions of the Association for Computational Linguistics*, *4*, 357-370.

CMAND. (2016). DFRWS Forensic Challenge 2016 from Center for Measurement and Analysis of Network Data (CMAND). Retrieved from http://old.dfrws.org/2016/challenge/index.shtml

European Commission. (2010). Standard on logging and monitoring.

Garfinkel, S. (2009). nps-2009-casper-rw: An ext3 file system from a bootable USB. Retrieved from http://downloads.digitalcorpora.org/corpora/drives/nps-2009-casper-rw/

Géron, A. (2017). *Hands-on machine learning with Scikit-Learn and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. O'Reilly Media, Inc.

Goller, C., & Kuchler, A. (1996). Learning task-dependent distributed representations by backpropagation through structure. *Neural Networks*, *1*, 347-352.

Graves, A., & Schmidhuber, J. (2005). Framewise phoneme classification with bidirectional LSTM networks. In *Proceedings of the IEEE International Joint Conference on Neural Networks* (pp. 2047-2052).

Gunestas, M., & Bilgin, Z. (2016). Log analysis using temporal logic and reconstruction approach: Web server case. *Journal of Digital Forensics, Security and Law*, *11*(2), 3.

He, P., Zhu, J., He, S., Li, J., & Lyu, M. R. (2016). An evaluation study on log parsing and its use in log mining. In *Proceedings of the 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks* (pp. 654-661).

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*(8), 1735-1780.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Lample, G., Ballesteros, M., Subramanian, S., Kawakami, K., & Dyer, C. (2016). Neural Architectures for Named Entity Recognition. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (pp. 260-270).

Lemoudden, M., Amar, M., & El Ouahidi, B. (2016). A binary-based mapreduce analysis for cloud logs. *Procedia Computer Science*, *83*, 1213-1218.

LogPAI Team. (2018). Loghub: A collection of system log datasets for massive log analysis. Retrieved from https://github.com/logpai/loghub

Marty, R., Chuvakin, A., & Tricaud, S. (2010). Challenge 5 of the Honeynet Project Forensic Challenge 2010 - Log Mysteries. Retrieved from http://honeynet.org/challenges/2010_5_log_mysteries

McGuire, P. (2007). Getting started with Pyparsing. O'Reilly Media.

Microsoft. (2005). LogParser: A powerful, versatile tool that provides universal query access to text-based data. Retrieved from https://www.microsoft.com/en-us/download/details.aspx?id=24659

Oliner, A., & Stearley, J. (2007). What supercomputers say: A study of five system logs. In *Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks* (pp. 575-584).

Parker, R., Graff, D., Kong, J., Chen, K., & Maeda, K. (2011). English Gigaword Fifth Edition LDC2011T07. DVD. Philadelphia: Linguistic Data Consortium.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., M. Blondel, Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*(Oct), 2825-2830.

Pennington, J., Socher, R., & Manning, C. (2014). GloVe: Global vectors for word representation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing* (pp. 1532-1543).

Rei, M. (2017). Semi-supervised Multitask Learning for Sequence Labeling. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (Vol. 1, pp. 2121-2130).

Rennie, J. D., Shih, L., Teevan, J., & Karger, D. R. (2003). Tackling the poor assumptions of naive Bayes text classifiers. In *Proceedings of the 20th International Conference on Machine Learning* (pp. 616-623).

Schatz, B., Mohay, G., & Clark, A. (2004). Rich event representation for computer forensics. In *Proceedings of the 5th Asia-Pacific Industrial Engineering and Management Systems Conference* (Vol. 2, No. 12, pp. 1-16).

Sconzo, M. (2018). SecRepo.com: Security data samples repository. Retrieved from http://www.secrepo.com/

Studiawan, H., Payne, C., & Sohel, F. (2017). Graph clustering and anomaly detection of access control log for forensic purposes. *Digital Investigation*, *21*, 76-87.

Tjong Kim Sang, E. F., & De Meulder, F. (2003). Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In *Proceedings of the 7th Conference on Natural Language Learning at HLT-NAACL* (pp. 142-147).

Tsuruoka, Y., Tsujii, J. I., & Ananiadou, S. (2009). Stochastic gradient descent training for l1-regularized log-linear models with cumulative penalty. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP* (pp. 477-485).

Wikimedia Foundation. (2014). English Wikipedia dumps. Retrieved from https://dumps.wikimedia.org/enwiki/