Black-box Attacks to Log-based Anomaly Detection

Shaohan Huang*, Yi Liu*, Carol Fung†, Hailong Yang*, Zhongzhi Luan*

*Sino-German Joint Software Institute, Beihang University, Beijing, China

†Concordia Institute for Information Systems Engineering, Concordia University, Montreal, Canada

{huangshaohan, yi.liu, luan.zhongzhi}@buaa.edu.cn

***Abstract*—Anomaly detection is the key to Quality of Service (QoS) in many modern systems. Logs, which record the run-time information of system, are widely used for anomaly detection. The security of the log-based anomaly detection has not been well investigated. In this paper, we conduct an empirical study on black-box attacks on log-based anomaly detection. We investigate eight different methods on log attacking and compare their performance on various log parsing methods and log anomaly detection models. We propose a method to evaluate the imperceptibility of log attacking methods. In our experiments, we evaluate the performance on the attack methods on two real log datasets. The results of our experiments show that LogBug outperforms the others in almost all situations. We also compare the imperceptibility of various attack methods and find a trade-off between performance and imperceptibility, where better attack performance means worse imperceptibility. To the best of our knowledge, this is the first work to investigate and compare the attack models on log-based anomaly detection.**

I. Introduction

Anomaly detection has been a critical task in the development of a reliable computer system. A good anomaly detection model can prevent high damage caused by anomalies, which is important for service management and system maintenance. In an operating system or other software system, logs are commonly used to record significant events and system status. System logs are one of the most important data sources for anomaly detection and system monitoring because they contain notable events and run-time status.

Log-based anomaly detection has been widely studied in previous research [1], [2]. Anomaly detection models usually take parsed logs as input and regard anomaly detection as a binary classification problem. Previous work has studied anomaly detection from different perspectives. Some studies aim at improving anomaly detection performance by employing novel machine learning or deep learning models [2], [3]. The goal of DeepLog [4] is to explore how to build an anomaly detection model from unlabeled data. LogRobust [1] is proposed to improve the robustness of anomaly detection models. However, the security of the log-based anomaly detection has not been well investigated yet. Some minor modifications to a part of the logs could change the semantics of raw log data and even completely deviate the returned results (e.g., misclassifying the anomalies, errors and failures of the systems). LogBug [5] is the first study to investigate log-related black-box attacks. It developed a real-time black-box attack approach to reduce the accuracy of log parser. However, it focuses solely on how to attack log parsers, ignoring the impact of log-based anomaly detection algorithms. Anomaly detection is a more practical task compared to log parsing, and it may be employed in real-world applications directly [1], [2]. In this work, we conduct a systematic review on the impact of various black-box attacks against log-based anomaly detection methods.

In this paper, we investigate eight different log attack methods. These log attack methods can be classified into three categories: word transformation, sequence transformation and LogBug. Word transformation and sequence transformation are inspired by text-related attacks [6]. LogBug is an adversarial logs generation method, proposed in [5]. We compare the performance of those attack methods on three log parsing approaches and five log-based anomaly detection models. We also provide a metric to measure how imperceptible log attack methods are.

In our experiments, we evaluated the performance of the attack methods on real log datasets including HDFS and Openstack. Our experimental results demonstrate that prefix transformation of LogBug achieves the best performance in different settings. We also evaluate the imperceptibility of different attack methods and find that better attack performance results in worse imperceptibility.

Our work provides a pipeline for generating a variety of log-related black-box attack methods that can be used to test the robustness of anomaly detection models. Furthermore, attacked log data can be used as an adversarial training data set to help improve anomaly detection models in the future [6].

The key contributions of this paper can be summarized as follows:

- This is the first work of the same kind that comprehensively evaluates the performance of attack models against log parsing and anomaly detection algorithms.
- We propose a metric to evaluate the imperceptibility of log attack methods.
- We provide a pipeline to evaluate the robustness of anomaly detection models.

978-3-903176-51-5 © 2022 IFIP

The rest of the paper is organized as follows. We introduce the background of our work in Section II. The architecture design and attack methods are described in Section III. Section IV describes the experimental settings, and metrics to evaluate the performance of attack methods. We present our experiment results in Section V. Related work is introduced in Section VI. Finally, in Section VII, we conclude our work and state possible future work.

II. Background

In this section, we will briefly introduce three log parsing and the log-based anomaly detection models that we will use in our evaluation.

A. Log Parsing

Modern systems often generate a large volume of logs to record run-time status. These logs are typically unstructured and free-textual, which need to be properly parsed before they are used for anomaly detection [7]. Log parsing is the process of converting unstructured messages into structured formats that include timestamps, log levels, templates, and parameters.

Many existing studies focus on log parsing [3], [8]–[13]. In this work, we select three common log parsing methods to evaluate the effectiveness of attacking methods. Most specifically, Spell [11] employs the longest common sub-sequence algorithm to parse logs in a streaming manner. Drain [12] applies a fixed-depth tree structure to parse log messages and extracts common templates. Paddy [13], an information retrieval based log parsing method, which uses a dynamic dictionary structure to speed up log parsing. Among these three methods, Paddy achieves the best performance compared to many other log parser methods [13]. These three methods are all online parsing methods and they do not require all log data to be available before parsing.

B. Log-based Anomaly Detection

Since system logs record noteworthy events and run-time status, they are one of the most important data sources for anomaly detection [2].

Existing log-based anomaly detection methods can be broadly classified into two categories based on the type of detection model: machine learning based and deep learning based. Statistical machine learning models (such as support vector machine and clustering) are used in machine learning-based methods to detect anomalies. These methods first extract log events from parsed logs and then convert log events into indexes feature spaces. For example, we assume that there are five types of log events in a corpus. One log template sequence is [E_1, E_2, E_5, E_2]. This sequence can be converted into one-hot vector $[1, 2, 0, 0, 1]$ as a feature vector. These vectors are used to train machine learning based detection model.

Deep learning models take a log stream as a natural language sequence that can take advantage of the semantic information in logs. For example, DeepLog [4] employs a stacked long short-term memory (LSTM) network [14] to encode log templates and then use another LSTM module to predict the next log. According to previous studies [1], [2], [4], anomaly detection methods based on deep learning outperform statistical machine learning models.

To investigate the performance of log attacks for different detection methods, we use three statistical machine learning models and two deep learning-based models in this paper.

III. Log Attacks

In this section, we first give an overview of log attacking, then introduce three log attacking methods, respectively.

A. Overview

We first give the problem definition of log attacks. We assume that $\mathcal{X}$ denotes system logs. A log parser can be defined as converter $\mathcal{F} : \mathcal{X} \rightarrow \mathcal{Y}$, where $\mathcal{Y}$ presents the set of log event templates. Log-based anomaly detection models take log templates $\mathcal{Y}$ and predict whether it is anomaly, which can be defined as $\mathcal{G} : \mathcal{Y} \rightarrow \mathcal{Z}$. For a log sequence $x \in \mathcal{X}$, anomaly detection model can predict correct label $z = \mathcal{G}(\mathcal{F}(x))$, where $z \in \mathcal{Z}$. A log attacker possibly modifies a log sequence x as $\hat{x}$, which deviates the anomaly detection model to generate a wrong label $\hat{z} = \mathcal{G}(\mathcal{F}(\hat{x}))$, where $z \neq \hat{z}$. The deviated accuracy of log-based anomaly detection model represents the effectiveness of log attacks and the similarity of x and $\hat{x}$ means the imperceptibility of attacks. We will introduce the details of evaluation methods in our experiment section.

The framework of log attacks is shown in Figure 1. Log-based anomaly detection systems first collect and save raw logs. We assume that anomaly detection systems are trained with non-attacked logs. A log attacker enters some sample logs and modifies some input logs when launching a log attack. The modified logs will perturb the log parsing methods (e.g., Spell [11]) and produce some wrong log templates. Anomaly detection models (e.g., SVM [15]) take the log templates containing some incorrect information as inputs. Finally, log attacks aim to reduce the precision and recall of anomaly detection model.

Log attacks can be classified into three categories: word transformation, sequence transformation and LogBug [5]. Word transformation modifies tokens in raw logs, such as randomly swapping some words in one log. Sequence transformation is to modify raw logs at sequence level. Log-based anomaly detection models usually use a sequence of logs as input and sequence-level information is also important for anomaly detection [2], [16]. Sequence transformation shuffles a sub-sequence in log sequence or randomly select an log event and repeat it several times in a log sequence. Sequence transformation does not modify the content of raw logs, so this method will not affect the result of log parsing. LogBug is an adversarial logs generation method, proposed in [5]. LogBug modifies the input logs using two different transformation methods: keyword and/or prefix transformation.

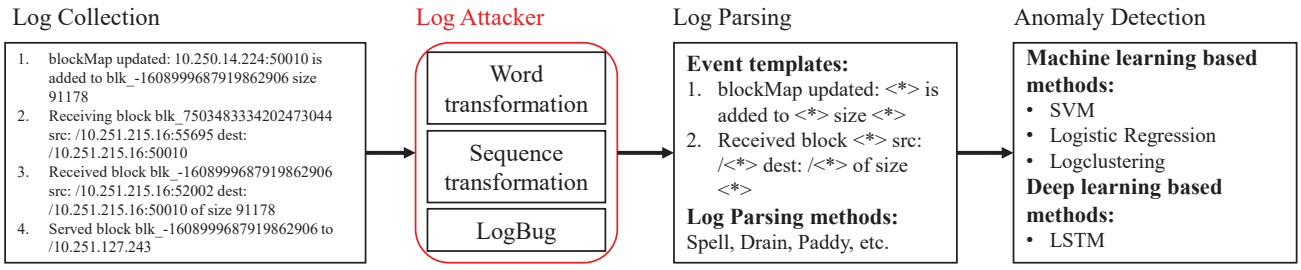

Fig. 1. Overview of black-box attacks to log-based anomaly detection.

B. Word Transformation

As introduced in [16], system logs are semi-structured texts "print"-ed by certain procedures of system and a log stream can be regarded as a natural language sequence. As a result, in a log-related black-box attack, text attack methods can be used.

TABLE I
WORD-LEVEL LOG ATTACK.

Original log:
Received block blk_a of size 67108864 from block blk_b
Swap in log:
Received block blk_a of **from** 67108864 **size** block blk_b
Drop in log:
Received ~~**block**~~ blk_a of size 67108864 from block ~~**blk_b**~~
Grammar error in log:
Reived block blk_a of size 67108864 from block **blk_cde**

We implement three common word-level attack methods in the word transformation part: swap, drop and grammar error. As illustrated in Table I, the original log is *Received block blk_a of size 67108864 from block blk_b*, which is from HDFS log dataset. Swap in log is to randomly swap two tokens in a raw log line. For example, we swap the token *from* and *size* in the original log. Drop in log is to randomly remove a few tokens from the original log message. In this case, we remove the token *block* and *blk_b*. Grammar error in log is to randomly select some tokens and inject spelling mistakes into them by removing or adding some content. As shown in Table I, we replace the token *Received* with *Reived* and *blk_b* with *blk_cde*.

Word transformation is a token-level attack on raw logs that can have a direct impact on log parsing results. For example, the original log in Table I can be parsed as a template *Received block * of size * from block **. If we use the drop method, the template would change to *Received * from block*. As a result, word transformation may produce a new parsed log or an unknown parsed log that is fed into anomaly detection models, causing the detection models' results to deviate or be misclassified.

C. Sequence Transformation

Log data is considered as natural time series data or natural language sequence. A sequence of logs is used as input in many log-based anomaly detection models. The importance of sequence-level information in anomaly detection has been demonstrated [2], [4]. Inspired by [2], who proposed a method to simulate unstable log sequences, we design three sequence-level log attack methods, namely deletion, insertion, and shuffle.

TABLE II
SEQUENCE-LEVEL LOG ATTACK

Original log sequence:
Log 1 → Log 2 → Log 3 → Log 4 → Log 5
Deletion in sequence:
Log 1 → Log 2 → ~~**Log 3**~~ → Log 4 → Log 5
Shuffle in sequence:
Log 1 → **Log 4** → **Log 2** → **Log 3** → Log 5
Insertion in sequence:
Log 1 → Log 2 → **Log 3** → **Log 3** → Log 4 → Log 5

As shown in Table II, the original log sequence is [*Log 1, Log 2, Log 3, Log 4, Log 5*]. Deletion in sequence is to randomly remove one log from the original log sequences. For example, delete *Log 3* to get the deviated log sequence. Shuffle in sequence is to shuffle a sub-sequence in log sequence. For example, we change [*Log 2, Log 3, Log 4*] to [*Log 4, Log 2, Log 3*]. Insertion in sequence means that we randomly select a log event and repeat it several times in a log sequence.

Since sequence transformation does not modify the content of raw logs, this method will not affect the result of log parsing. Sequence transformation, on the other hand, affects sequence-level information and results in unstable log data [2].

D. LogBug

LogBug can universally perturb different system logs without the knowledge of log parser [5]. LogBug first denotes a function $\tau(x)$ to evaluate the risk degree of each log x. Assume that a log x has a risk degree, indicating how critical the log x is to the system. LogBug aims at reducing $\tau(x)$ to a lower risk degree by deviating the log x. To increase the

stealthiness of the attack, it will modify the current log x to $\hat{x}$ by minimizing the Euclidean distance $||v(x) - v(\hat{x})||$, where $v(x)$ is the TF-IDF vector of x.

As illustrated in [5], LogBug modifies the input logs using two different transformation methods: keyword and/or prefix transformation. Keyword transformation generates candidate adversarial logs by recursively adding, removing or replacing the keywords in x. It's similar to the word transformation, but the goal is to reduce the degree of risk and minimize the distance. Because of the objective limitation, keyword transformation in LogBug has less randomness than word transformation approaches.

Prefix transformation performs more efficient attack with slightly more modifications. It modifies the prefix of the logs to generate the adversarial logs with low-risk event templates. First, the attacker uses their previously collected event templates as a database, prefix transformation locates and/or replace the prefixes by using the database. Prefix transformation, however, may diverge the log contents more than keyword transformation if the prefix tree entries are very different.

IV. Experiment Setting

A. Datasets

We conduct our experiments on two real log datasets: the HDFS dataset [17] and the OpenStack dataset [4]. The detailed information on the two datasets are described as follows:

(1) HDFS: The HDFS dataset is collected through running Hadoop-based jobs on more than two hundred Amazon's EC2 nodes. It consists of 11,175,629 logs. Developers manually labeled anomalies through handcrafted rules. Each HDFS log contains a unique block ID for each block operation. We use block ID as identifier to build log sequences. There are 575,061 blocks of logs in the dataset, among which 16,838 blocks were labeled as anomalous.

(2) OpenStack: OpenStack dataset is generated on CloudLab, which is a cloud operating system including large pools of computer, storage, and networking resources. It contains 70,746 INFO level logs and three types of anomalies. OpenStack data is grouped into different sessions by instance ID. There are 503 instances labeled as anomalous.

In the following experiments, we leverage the front 80% (according to the timestamps of logs) as the training data, and the remaining 20% as the testing data. In our log attacking experiments, we assume that log-based anomaly detection models are trained on non-attacked data. As a result, we train our anomaly detection models using non-attacked training data and then test the dataset using various log attacking methods before evaluating these attack methods. Moreover, because the above datasets were manually labeled, we take these labels as the ground truth for evaluation.

B. Evaluation Metrics

We evaluate log attacks from two perspectives: effectiveness of attacks and imperceptibility of attacks.

1) Reduced F1-score: We use reduced F1-score to evaluate the effectiveness of log attacks. Anomaly detection is a binary classification task, where F1-score is an important metric to evaluate detection models. F1-score is the harmonic average of Precision and Recall. We use F1-score to stands for the performance of anomaly detection model. Therefore, reduced F1-score is used to evaluate the effectiveness of log attacks.

2) Imperceptibility: Log data, unlike image data or general text data, is a type of time series data. The log attacker's imperceptibility includes not only the differences at the single log message level, but also the changes at the log sequence level.

We use Levenshtein edit distance [18] to evaluate the dissimilarity between the original log and the attacked log from both the single log message level and the log sequence level. Edit distance is a metric to quantify the dissimilarity between two strings by counting the minimum number of operations required to transform one string into the other.

We compute imperceptibility from both single log message level and log sequence level. For example, the original log sequence is S_1 and the attacked log sequence is S_2, where $S_1 = \{l_1, l_2, ..., l_n\}$ and $S_2 = \{l'_1, l'_2, ..., l'_n\}$. The imperceptibility of S_1 and S_2 is computed as follows:

$$Imperceptibility = \frac{1}{2}(\frac{1}{n}\sum_{k=1}^{n} Dis(l_k, l'_k) + Dis(S_1, S_2)) \quad (1)$$

where Dis represents the Levenshtein edit distance. The minimum value of imperceptibility is 0 and its maximum value is 1. Smaller imperceptibility values mean that the attacked logs are more similar to original logs and harder to detect. In order to normalize the imperceptibility metric into 0-1, we add 1/2 as weight.

C. Implementation

We investigate three log parsing methods and five anomaly detection models including three machine learning based models and two deep learning based models. These log parsing methods are briefly described as follows:

- Spell [11]: Spell employs the longest common subsequence algorithm to parse logs in a streaming manner.
- Drain [12]: Drain applies a fixed-depth tree structure to parse log messages and extracts common templates.
- Paddy [13]: Paddy is an information retrieval based log parsing method, which designs a dynamic dictionary structure to improve the performance of log parsing.

The anomaly detection methods used in this paper are listed as follows, where LogCluster, SVM and LR are machine learning based models and DeepLog and LogRobust belong to deep learning based models.

- LogCluster (LC) [15]: LogCluster takes anomaly detection as a clustering problem. It employs a knowledge base to reduce redundancy by previously examining log sequences.

TABLE III
EVALUATION ON HDFS DATA SET. WT MEANS WORD TRANSFORMATION, ST MEANS SEQUENCE TRANSFORMATION, GE PRESENTS GRAMMAR ERROR.

		LC	SVM	LR	DL	LogR	Ave.
Ori. F1		0.91	0.95	0.78	0.96	0.97	-
WT	Wrap	0.84	0.89	0.65	0.89	0.95	0.07
	Drop	0.90	0.94	0.71	0.91	0.96	0.03
	GE	0.85	0.88	0.68	**0.86**	0.95	0.07
ST	Delete	0.88	0.93	0.74	0.91	0.96	0.03
	Shuffle	0.91	0.95	0.78	0.91	0.93	0.018
	Insert	0.87	0.93	0.73	0.93	0.95	0.032
LogBug	Keyword	0.78	0.74	0.63	0.91	0.93	0.116
	Prefix	**0.72**	**0.69**	**0.62**	**0.86**	**0.89**	**0.158**

TABLE IV
EVALUATION ON OPENSTACK DATA SET. WT MEANS WORD TRANSFORMATION, ST MEANS SEQUENCE TRANSFORMATION, GE PRESENTS GRAMMAR ERROR.

		LC	SVM	LR	DL	LogR	Ave.
Ori. F1		0.42	0.64	0.62	0.95	0.82	-
WT	Wrap	0.38	0.57	0.55	0.87	0.74	0.068
	Drop	0.41	0.61	0.57	0.89	0.79	0.036
	GE	0.38	0.56	0.53	0.83	0.75	0.08
ST	Delete	0.40	0.61	0.58	0.92	0.79	0.03
	Shuffle	0.42	0.64	0.62	0.91	0.77	0.018
	Insert	0.39	0.61	0.57	0.91	0.79	0.036
LogBug	Keyword	0.38	0.49	0.57	0.87	0.77	0.074
	Prefix	**0.34**	**0.44**	**0.48**	**0.82**	**0.72**	**0.13**

- SVM [7]: Support vector machine (SVM) is a classification algorithm, which first converts log sequences to count vectors and then learns a set of support vectors to detect outlier.
- LR [19]: Logistic regression (LR) is similar to SVM [7]. It also uses count vectors as input and then applies LR as its supervised learning algorithm.
- DeepLog (DL) [4]: DeepLog is a deep neural network model using LSTM to model a system log as a natural language sequence.
- LogRobust (LogR) [1]: LogRobust extracts semantic information of log events and represents them as semantic vectors. It utilizes an attention-based Bi-LSTM model to detect anomalous log sequences.

For word transformation and sequence transformation attacks, we randomly select 5% log of the total test data to attack. We wrap or drop words in log record only once to make those operations be compared fairly. We utilize LogBug to create modified test data in which all the log lines with level 'WARN' are deviated. The modified log lines account for only 6% of the total test data.

V. RESULTS

A. Attacks to Different Detection Models

In this part, we evaluate the performance of the eight types of black-box attacks on different anomaly detection methods on the HDFS and Openstack log data, where we use Drain [12] as our log parser.

Table III shows the performance of different log attack methods over the HDFS data set. The first row is the original F1-scores and the others are the F1-scores of attacked models. We can see that the original F1-scores are very high (i.e., more than 90%) for all five anomaly detection methods on HDFS log dataset. Compared to the original results, prefix transformation of LogBug can drastically reduce by 19% for LogCluster and 26% for SVM. In an average setting, prefix transformation can reduce the F1-scores of different log anomaly detection models by 15.8% points. The results confirm that black-box attacks can degrade the performance of anomaly detection models significantly.

Furthermore, the reduced F1-scores of all attack methods are not the same. Prefix transformation of LogBug achieves the highest reduction on all attack models and keyword transformation of LogBug also performs better than word transformation and sequence transformation. It is because that word transformation and sequence transformation randomly select the log of the test data to attack. On contract, LogBug attacks log lines with level 'WARN'. This finding shows that the 'WARN' level log lines are more important for anomaly detection. We also observe that sequence transformation is the weakest attack type. The first reason is that LogCluster, SVM and LR models use count vectors as their features and these features are order independent. Therefore, shuffling has no impact on the performance of these models. The second reason is that word transformation or LogBug is to modify raw logs at word level and they both affect the result of log parsing and generate new log templates. Sequence transformation does not modify the content of raw logs and only affects the sequence-level information. The results demonstrate that anomaly detection models rely more on word-level information than sequence-level information.

We also conducted our experiments on the Openstack data set and the results are shown in Table IV. Similar to the Table III, the last row is the original F1-scores and others are the reduced F1-scores. We can see that prefix transformation of LogBug has the highest impact on F1-score reduction on all detection models. It also implies that 'WARN' level log is important for Openstack dataset. How to find vulnerable logs to attack is an interesting research direction in the future.

Some additional insights can be derived from word transformation attack. Wrap and GrammarError performs better than drop in log since drop in log may remove some unimportant tokens from log and does not impact the result of log parsing. On the Openstack dataset, the sequence transformation is also the weakest attack type compared to word transformation and LogBug, which confirms that word-level information is more important than sequence-level in Openstack dataset.

B. Attacks to Different Log Parsers

In this section, we compare the performance of black-box attacks to different log parser methods. We conduct our

TABLE V
EVALUATION ON HDFS DATA SET WITH DIFFERENT LOG PARSERS.

		Spell	Drain	Paddy	Aver.
Original F1		0.90	0.97	0.95	-
Word Trans.	Wrap	0.85	0.95	0.94	0.03
	Drop	0.89	0.96	0.95	0.01
	GrammarError	0.83	0.95	0.93	0.04
LogBug	Keyword	0.81	0.93	0.92	0.05
	Prefix	**0.75**	**0.89**	**0.84**	**0.11**
Average Drop		0.07	0.03	0.03	-

TABLE VI
IMPERCEPTIBILITY ON HDFS DATA SET. ATTACKER WITH SMALLER IMPERCEPTIBILITY VALUE IS HARDER TO DETECT.

Word Transformation			Sequence Transformation			LogBug	
Wrap	Drop	GE	Delete	Shufle	Insert	Keyword	Prefix
0.32	0.25	0.27	0.14	0.19	**0.11**	0.37	0.41

experiments on the HDFS dataset and use LogRobust as our anomaly detection model. Since sequence transformation methods do not affect the result of log parsing, we only compare word transformation and LogBug.

We can observe from Table V that prefix transformation of LogBug achieves the highest F1-score reduction with different log parsers and LogBug outperforms than word transformation. It also shows that prefix transformation of LogBug is the most effective attack model on log-based anomaly detection algorithms. Several additional insights can be learned from the next-to-last row. The reduced F1-score of all the log parsers are not the same since their algorithms would generate slightly different results on the same deviated logs. We can find that Spell is the most vulnerable to attack compared to Drain or Paddy. It it because Spell uses the longest common subsequence algorithm to parse logs and it cannot find a proper sub-sequence in deviated logs. The result reveals that Drain and Paddy are robust against log data attack.

C. Imperceptibility

In this section, we compare the imperceptibility of different log attack methods. As mentioned in Section IV-B2, imperceptibility is an important evaluation metric for log attackers. Imperceptibility means that the modified logs are closer to the original logs and difficult to identify. Different from image data or general text data, we propose a new method to compute imperceptibility from both single log message level and log sequence level. We only compute the imperceptibility of attacked log sequences.

As shown in Table VI, we evaluate the imperceptibility of different log attack methods. Smaller imperceptibility values mean that attacked logs are more similar to original logs and harder to detect. We can observe that insert of sequence transformation method achieves the best imperceptibility over these attack methods, it is because that sequence transformation methods do not modify the content of raw log and only affect imperceptibility at the sequence level. Table VI shows that deviated logs generated from LogBug have lower similarity than other methods, which means that the LogBug attack is easier to be detected. Some additional insights can be derived from the relationship between the attack performance and the imperceptibility. Better attack performance results in worse imperceptibility, indicating a trade-off between the effectiveness and the concealability.

VI. RELATED WORK

System logs can be used to record the runtime status or events in almost all computer systems. There have been many studies on log-based anomaly detection [4], [7], [20]–[22]. We have introduced the background of log-based anomaly detection in Section II-B. The robustness of log-based anomaly detection models is critical. We choose this problem not only because it is challenging, but also because anomaly detection models are widely used in many safety and security sensitive applications, e.g., online computing platform [17] or cloud service [4].

The first study to investigate log-related black-box attacks is LogBug [5]. The authors developed a real-time black-box attack approach for deviating log parser detection accuracy by gently altering logs without knowing the log parser's learning model and settings. LogBug, on the other hand, only considered how to attack log parsers and ignored the influence of log-based anomaly detection models. Compared to log parsing, anomaly detection is a more practical task and it can be used in real-world scenarios directly [20], [21], [23]. Therefore, our work systematically analyzes the impact of different black-box attacks to various log-based anomaly detection methods.

Recently existing automatic log anomaly detection approaches [16] regard a log stream as a natural language sequence and leverage some natural language processing methods to improve log anomaly detection. Therefore, the intuitions and methods in text related attack methods can be applied in log attack. Many algorithms can generate adversarial texts to perturb the learning models [6]. [24] addressed three attack types: dropping, adding, and swapping internal characters within words. [25] proposed a word deletion method with greedy search. [26] introduced a word-level attack method, which incorporates the semantic-based word substitution method and particle swarm optimization-based search algorithm. Text attack is not the focus of our work, therefore we choose three common word-level attack methods in our experiments. Referring to log structure's features, we propose a sequence-level log attack method, which will shuffle a sub-sequence or randomly repeat a log event.

Black-box attacks not only help understand the vulnerabilities and security of log-based anomaly detection systems but also can be used to generate adversarial data to improve the performance of detection model [6]. In [1] and [2], they introduced how to synthesize the unstable log sequences data to evaluate the robustness of anomaly detection models. More diversity log data can be generated as a result of our

study, which can be used as training data to improve model robustness.

VII. Conclusion

In this paper we take the first step to study black-box attacks to log-based anomaly detection models. We investigate eight different log attack methods including word-level transformation and sequence-level transformation. We compare the performance of those attack methods on different log parsing approaches and log-based anomaly detection models. Our experimental results demonstrate that prefix transformation of LogBug achieves the best performance in various settings and sequence transformation is the weakest attack method. We also propose a metric to evaluate the imperceptibility of log attack methods. Our experiment demonstrates there is a trade-off where better attack performance results in worse imperceptibility.

One of the future directions of our work is to utilize the attacked log data to train log parsers and log-based anomaly detection models. This will be able to improve the robustness of anomaly detection models and help models models better defend against attacks.

Acknowledgment

This work was supported by National Natural Science Foundation of China (No. 62072018)

References

[1] X. Zhang, Y. Xu, Q. Lin, B. Qiao, H. Zhang, Y. Dang, C. Xie, X. Yang, Q. Cheng, Z. Li *et al.*, "Robust log-based anomaly detection on unstable log data," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, pp. 807–817.

[2] S. Huang, Y. Liu, C. Fung, H. Yang, and Z. Luan, "Hitanomaly: Hierarchical transformers for anomaly detection in system log," *IEEE Transactions on Network and Service Management*, vol. 17, no. 4, pp. 2064–2076, 2020.

[3] P. He, J. Zhu, S. He, J. Li, and M. R. Lyu, "An evaluation study on log parsing and its use in log mining," in *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2016, pp. 654–661.

[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*, 2017, pp. 1285–1298.

[5] J. Sun, B. Liu, and Y. Hong, "Logbug: Generating adversarial system logs in real time," in *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, 2020, pp. 2229–2232.

[6] J. X. Morris, E. Lifland, J. Y. Yoo, and Y. Qi, "Textattack: A framework for adversarial attacks in natural language processing," *arXiv preprint arXiv:2005.05909*, 2020.

[7] P. He, J. Zhu, S. He, J. Li, and M. R. Lyu, "Towards automated log parsing for large-scale log data analysis," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 6, pp. 931–944, 2017.

[8] K. Yamanishi and Y. Maruyama, "Dynamic syslog mining for network failure monitoring," in *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*. ACM, 2005, pp. 499–508.

[9] A. Oliner, A. Ganapathi, and W. Xu, "Advances and challenges in log analysis," *Communications of the ACM*, vol. 55, no. 2, pp. 55–61, 2012.

[10] S. Niwattanakul, J. Singthongchai, E. Naenudorn, and S. Wanapu, "Using of jaccard coefficient for keywords similarity," in *Proceedings of the international multiconference of engineers and computer scientists*, vol. 1, no. 6, 2013, pp. 380–384.

[11] M. Du and F. Li, "Spell: Streaming parsing of system event logs," in *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE, 2016, pp. 859–864.

[12] 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)*. IEEE, 2017, pp. 33–40.

[13] S. Huang, Y. Liu, C. Fung, R. He, Y. Zhao, H. Yang, and Z. Luan, "Paddy: An event log parsing approach using dynamic dictionary," in *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2020, pp. 1–8.

[14] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with lstm," 1999.

[15] Q. Lin, H. Zhang, J.-G. Lou, Y. Zhang, and X. Chen, "Log clustering based problem identification for online service systems," in *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*. IEEE, 2016, pp. 102–111.

[16] W. Meng, Y. Liu, Y. Zhu, S. Zhang, D. Pei, Y. Liu, Y. Chen, R. Zhang, S. Tao, P. Sun *et al.*, "Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19. International Joint Conferences on Artificial Intelligence Organization*, vol. 7, 2019, pp. 4739–4745.

[17] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, "Detecting large-scale system problems by mining console logs," in *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, 2009, pp. 117–132.

[18] L. Yujian and L. Bo, "A normalized levenshtein distance metric," *IEEE transactions on pattern analysis and machine intelligence*, vol. 29, no. 6, pp. 1091–1095, 2007.

[19] Y. Liang, Y. Zhang, H. Xiong, and R. Sahoo, "Failure prediction in ibm bluegene/l event logs," in *Seventh IEEE International Conference on Data Mining (ICDM 2007)*. IEEE, 2007, pp. 583–588.

[20] M. Cinque, D. Cotroneo, and A. Pecchia, "Event logs for the analysis of software failures: A rule-based approach," *IEEE Transactions on Software Engineering*, vol. 39, no. 6, pp. 806–821, 2012.

[21] A. Oprea, Z. Li, T.-F. Yen, S. H. Chin, and S. Alrwais, "Detection of early-stage enterprise infection by mining large-scale log data," in *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE, 2015, pp. 45–56.

[22] T. Jia, P. Chen, L. Yang, Y. Li, F. Meng, and J. Xu, "An approach for anomaly diagnosis based on hybrid graph model with logs for distributed services," in *2017 IEEE International Conference on Web Services (ICWS)*. IEEE, 2017, pp. 25–32.

[23] J. P. Rouillard, "Real-time log file analysis using the simple event correlator (sec)." in *LISA*, vol. 4, 2004, pp. 133–150.

[24] D. Pruthi, B. Dhingra, and Z. C. Lipton, "Combating adversarial misspellings with robust word recognition," 2019.

[25] S. Feng, E. Wallace, A. G. I. au2, M. Iyyer, P. Rodriguez, and J. Boyd-Graber, "Pathologies of neural models make interpretations difficult," 2018.

[26] Y. Zang, F. Qi, C. Yang, Z. Liu, M. Zhang, Q. Liu, and M. Sun, "Word-level textual adversarial attacking as combinatorial optimization," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online: Association for Computational Linguistics, Jul. 2020, pp. 6066–6080. [Online]. Available: https://aclanthology.org/2020.acl-main.540