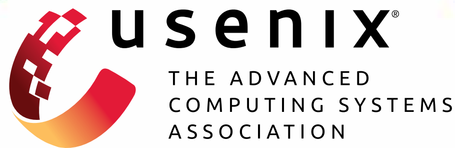


AirTag: Towards Automated Attack Investigation by Unsupervised Learning with Log Texts

Hailun Ding, *Rutgers University;* Juan Zhai, *University of Massachusetts Amherst;*
Yuhong Nan, *Sun Yat-sen University;* Shiqing Ma, *University of Massachusetts Amherst*

https://www.usenix.org/conference/usenixsecurity23/presentation/ding-hailun-airtag

This paper is included in the Proceedings of the
32nd USENIX Security Symposium.

August 9–11, 2023 • Anaheim, CA, USA

978-1-939133-37-3

Open access to the Proceedings of the
32nd USENIX Security Symposium
is sponsored by USENIX.

AIRTAG: Towards Automated Attack Investigation by Unsupervised Learning with Log Texts

Hailun Ding
Rutgers University

Juan Zhai
University of Massachusetts Amherst

Yuhong Nan
Sun Yat-sen University

Shiqing Ma
University of Massachusetts Amherst

Abstract

The success of deep learning (DL) techniques has led to their adoption in many fields, including attack investigation, which aims to recover the whole attack story from logged system provenance by analyzing the causality of system objects and subjects. Existing DL-based techniques, e.g., state-of-the-art one ATLAS, follow the design of traditional forensics analysis pipelines. They train a DL model with labeled causal graphs during offline training to learn benign and malicious patterns. During attack investigation, they first convert the log data to causal graphs and leverage the trained DL model to determine if an entity is part of the whole attack chain or not. This design does not fully release the power of DL. Existing works like BERT have demonstrated the superiority of leveraging unsupervised pre-trained models, achieving state-of-the-art results without costly and error-prone data labeling. Prior DL-based attacks investigation has overlooked this opportunity. Moreover, generating and operating the graphs are time-consuming and not necessary. Based on our study, these operations take around 96% of the total analysis time, resulting in low efficiency. In addition, abstracting individual log entries to graph nodes and edges makes the analysis more coarse-grained, leading to inaccurate and unstable results. We argue that log texts provide the same information as causal graphs but are fine-grained and easier to analyze.

This paper presents AIRTAG, a novel attack investigation system. It is powered by unsupervised learning with log texts. Instead of training on labeled graphs, AIRTAG leverages unsupervised learning to train a DL model on the log texts. Thus, we do not require the heavyweight and error-prone process of manually labeling logs. During the investigation, the DL model directly takes log files as inputs and predicts entities related to the attack. We evaluated AIRTAG on 19 scenarios, including single-host and multi-host attacks. Our results show the superior efficiency and effectiveness of AIRTAG compared to existing solutions. By removing graph generation and operations, AIRTAG is 2.5x faster than the state-of-the-art method, ATLAS, with 9.0% fewer false positives and 16.5% more true positives on average.

1 Introduction

Deep Learning (DL) has achieved state-of-the-art results in many Artificial Intelligence (AI) tasks, e.g., image classification and natural language understanding [9, 16, 31, 36]. Recent advances in pre-trained models further advanced the field [10, 54]. For example, BERT [10] is a pre-training technique for natural language understanding. It trains a bidirectional transformer model on a large corpus of unlabeled texts and fine-tunes the pre-trained model for downstream tasks on a small labeled dataset. This technique has achieved superior results in over ten downstream tasks than prior methods. DL techniques can learn patterns that are not obvious to humans from massive data without traditional feature engineering, which helps identify irregular patterns and automate processes requiring analyzing a large amount of data. Researchers then propose to leverage such capabilities of DL in security analysis, in which administrators face bulk data, and the suspicious ones take a small percentage of it. DL-based techniques have achieved promising results in many security applications, e.g., malware detection [22], binary analysis [20, 37, 60, 70], and network traffic analysis [43, 46, 55].

Attack investigation is a typical security analysis in forensics analysis, aiming to recover an attack chain (or attack story) from massive log data from different system components (e.g., operating system, network devices). Starting from given symptom events, a common practice of performing the investigation is to scan the provenance information and analyze the causal relationships among all objects and subjects. Notice that the whole attack chain can contain normal system operation events which are hard to be detected by anomaly detection techniques [19, 35]. The procedure of attack investigation is extremely labor-intensive as it requires analyzing the wealth of data because of the large numbers of affected components, long duration of attacks, and complex behaviors of modern systems. Prior works have shown that modern computing systems generate gigabytes of data even for a single desktop per day [23, 34, 67]. Complex attacks involving advanced attack skills, multiple organizations, and longer at-

tack periods, make the investigation more challenging. For example, the 2020 U.S. federal government data breach lasted months, affecting tens of U.S. federal, state, and local governments and private sectors [14]. As a typical example of Advanced Persistent Threats (APTs), it exploited vulnerabilities in Microsoft, SolarWinds, and VMware products, services, and software distribution infrastructures. The attack pattern and causality in such large data are not obvious. Researchers proposed techniques to automate the investigation and better identify hidden patterns by leveraging AI techniques. According to the report from IBM [27], organizations that deploy AI techniques can identify and control attacks 28 days faster, saving $3.05 million in costs.

1.1 Existing Solution and Motivation

Prior research has proposed learning-based approaches to boost the analysis process and reduce human burden. The intuition is that despite the payload executed and vulnerabilities exploited, cyberattacks share common attack strategies that DL models can learn. ATLAS [3] is state-of-the-art Deep Learning (DL) based attack investigation framework. The basic idea of ATLAS is to learn from labeled benign and malicious logs to identify benign and malicious attack patterns. In the following, we will use an example to show how ATLAS works and its limitation.

Example. In a spam campaign attack, the adversary started the attack by spreading phishing links via social engineering. A victim user, Alice, clicked the malicious link. `Firefox_17` resolved the domain name by using DNS server `192.b.c.d` and opened the website `xaa.com`. Then, Alice downloaded a file `msf.rtf` and opened it with MS Word (`winword_7`). Unfortunately, `msf.rtf` was a malicious file that exploited the CVE-2017-11882 vulnerability and ran a malware `payload` on Alice's machine. This malware scanned the whole disk to look for PDF files and then sent them to the attacker. The malicious activities happened in the background, mixing with benign activities of `winword`, `Firefox`, etc. Later, the network security tool detected the malicious website `xaa.com`. This is a simplified real-world attack that accounted for nearly three-quarters of all exploits in 2020 Q4 and affected multiple countries including the U.S., Australia, and Japan [26].

Investigation process of ATLAS. The investigation in this context aims to locate all attack entities and how they lead to the payload by starting from the detected website `xaa.com`. Component A in Figure 1 shows the process of ATLAS and the investigating results of the example.

For a given log file and symptom events, ATLAS first constructs the causalities as a causal graph and reduces the graph size with additional expert knowledge, e.g., deleting repeated edges to facilitate its future analysis (Figure 1-A.1). In this graph, nodes are entities such as system objects (e.g., files, sockets) and system subjects (i.e., processes), and edges are the causal relations between entities (e.g., read, write). Then, ATLAS iteratively classifies other nodes in the graph as either malicious or benign, leveraging given symptom events. Specifically, the classification of other nodes is based on a deep-learning model which learns the correlated patterns between different attack nodes in the graph. In this example, given `xaa.com` as the initial attack clue, ATLAS detects the compositions of all nodes with the attack clue `xaa.com` and determines whether they are part of the attack chain (Figure 1-A.2). Once a new attack node `Payload_1` is detected, ATLAS adds this node to the attack node list and continually detects compositions of more attack nodes with other vertices in the graph. Lastly, the attack story is recovered as a set of attack nodes (e.g., `xaa.com`, `Payload_1`, `Payload.exe`, and `149.a.b.c`), as well as nodes and edges which are directly connected with them (Figure 1-A.3).

The core idea of learning-based methods like ATLAS is to automatically capture inherent information of provenance graphs to differentiate the benign and attack behaviors, reducing human efforts to reconstruct the attack story. Despite its progress in automating the whole attack investigation process, existing work suffers from the following limitations.

Limitation I: extensive manual efforts. Existing learning-based methods are supervised learning methods, requiring extensive manual efforts to label the training data, which are labor-intensive, costly, and error-prone. Unlike many AI tasks which have publicly available training data, there is no well-labeled dataset for attack forensics. Attack investigation works on detailed logs containing confidential information about organizations, their infrastructures, and operations. Due to privacy and security concerns, obtaining publicly available labeled logs is challenging. Existing studies have shown that computer vision datasets contain many label errors [7, 58, 64], introducing noises in training data and affecting the model performance. ATLAS requires labels of all nodes as `malicious` or `benign`, and trains a classifier on labeled data to classify node compositions. Labeling datasets like this requires knowledge and understanding of the logs and cybersecurity, which is costly and error-prone.

Limitation II: high computing costs. As shown by existing work [39], fast investigation can shorten the time of fixing compromised systems, significantly reduce financial loss, and help understand attack intentions to prevent potential future damages. Existing methods learn behavioral patterns from causal graphs, and graph construction and operations (e.g., search, traversal) have a high overhead. As shown in Figure 1-A, ATLAS first converts logs to graphs at the beginning of forensics analysis. After detecting all attack nodes, ATLAS reconstructs the attack story by connecting involved nodes (Figure 1-A.3). Generating such graphs is the bottleneck of this analysis, taking over 96% of the total time (see §3). On the other hand, processing graphs is a well-known complex problem, and using it in attack investigation leads to high computing costs [39]. Notice that logs and graphs (with detailed annotations) are two different representations of the same

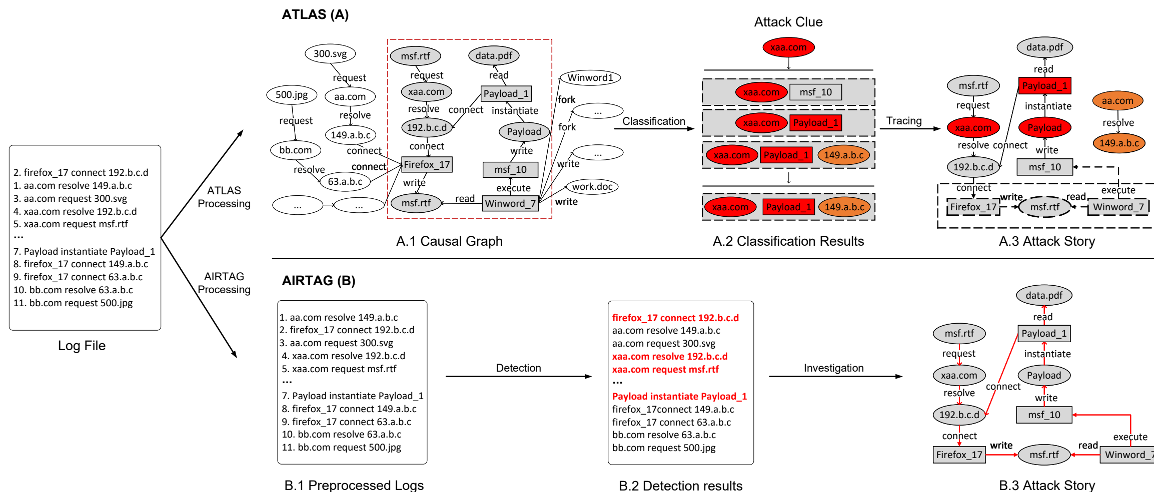


Figure 1: Comparison of AIRTAG and ATLAS. (Red nodes/edges: correctly detected malicious entities; Orange nodes: false positives; Bold and dashed box: missed malicious entries by ATLAS). ATLAS first preprocesses logs and converts logs into causal graphs (A.1). Then, it iteratively classifies each node in the causal graph and marks them as benign or malicious (A.2). Finally, ATLAS reconstructs the attack story by finding all activities related to malicious nodes (A.3). For AIRTAG, we first preprocesses logs (B.1) and then directly detects malicious entries (B.2). Finally, AIRTAG recovers the attack story (B.3).

information, which are convertible to each other. Graphs are more accessible for humans to inspect and digest but hard for computers to process. Texts are in Euclidean space, which is easier for computers to process, while graphs are not.

Limitation III: inaccurate and unstable coarse-grained analysis. Existing investigation solutions leverage causal graphs during the investigation, which are *abstracted*. Such abstracted causal graphs help facilitate human inspection, but there is a mismatch between these graph primitives and log events. In these graphs, a node represents an entity, e.g., a process `Firefox`, while in the log file, a single `Firefox` entry is an execution state of this process. The execution state transitions can be triggered by inner logic (e.g., timer events) or interactions with other processes (e.g., user inputs, inter-process communications). The `Firefox` node in causal graphs is a set of these execution states. Similarly, a user can change the entire file without changing its `inode` number and path. These abstract nodes omit the details of program execution states that are hard to present and are more accessible to humans. Using such abstract graph primitives is coarse-grained, leading to inaccurate (i.e., higher false positive/negative rates) and unstable results (i.e., dramatically different results when the symptom events are different). Existing work [45] has observed this. For the motivating example in Figure 1, when using `xaa.com` as the symptom event, ATLAS misses the node `Firefox` and all related activities, leading to a low true positive rate. Meanwhile, ATLAS misclassifies the node `149.a.b.c` (represented by the orange color), causing high false positive rate. Our further in-depth analysis also confirms this observation (§3.4).

1.2 Our Solution

This paper proposes AIRTAG, an unsupervised learning-based attack investigation method that works on raw log texts rather than causal graphs. We argue that using causal graphs during the investigation is inaccurate, coarse-grained, and expensive. Instead, we propose to use log entries (i.e., texts) during analysis and convert them to causal graphs when needed (i.e., presenting to humans as the final result). Using raw log entries in the analysis also enables us to leverage state-of-the-art unsupervised learning methods, avoiding manual labeling. We use Natural Language Processing (NLP) techniques to process logs because log data is language-like data and NLP techniques can capture text semantics most effectively. Many existing works [3, 11, 12] also use NLP for log-related security tasks, and the performance is proven to be good.

As illustrated in Figure 1-B, AIRTAG takes the log file and symptom events as input, processes the log file (Figure 1-B.1, like NLP tokenization), classifies each log entry to `benign` or `malicious` (Figure 1-B.2), and then reconstruct the attack story (Figure 1-B.3). The classifier for identifying attack-relevant log entries is trained with unsupervised learning methods that do not require manual effort for data labeling. Since AIRTAG performs analysis at a more fine-grained level (i.e., log entry), this greatly improves the accuracy and robustness (against changes of symptom events) for attack investigation. In addition, AIRTAG is also more efficient as it removes the time-consuming graph construction and operations. Similar to NLP tasks, AIRTAG first preprocesses the log texts (e.g., tokenization and embedding) and then predicts a given log entry as `benign` or `malicious` (red in B.2 of Figure 1). These entries can reconstruct the attack story: `Firefox` connects to

the malicious website `192.b.c.d`, DNS server resolves the address, `Firefox` requests the file `msf.rtf`, and the malware starts the malicious `Payload` process. The results completely and honestly recover the attack story. Applying unsupervised learning on raw log data for attack detection is non-trivial.

Challenge I: One challenge is that logs have domain-specific syntaxes and semantics. Logs are similar to natural language artifacts but different. As logs are designed for humans to read and understand, they mainly contain natural language words. Traditional NLP methods all use a large dictionary containing known words, and the number of unknown words is small (compared with the number of known words). They also leverage predefined rules (e.g., convert the word `recurringly` to two tokens `[recurring]` and `[##ly]`) to reflect natural language features, e.g., part-of-speech (POS). When training embeddings for these preprocessed tokens, models can leverage the syntaxes and semantics (e.g., tense) to improve performance. We can leverage them to process known words in logs. On the other hand, logs unavoidably use domain-specific or application-specific words or symbols. For example, a lot of log entries like paths and names of directories are sequences of words organized in a specific format (e.g., Linux uses slash to separate directories in paths and reflect their hierarchical structure). NLP tokenization and embedding techniques cannot recognize and leverage such syntaxes and semantics. Also, many programs and files are not using common English words as names. Existing techniques will identify these log texts (e.g., file names, paths) as unknown words. Consequently, these words will have the same token, i.e., `UNKNOWN`, and the same token embedding. If so, the number of unknown words will be significant, resulting in impaired performance.

We solve this challenge by applying domain-specific rules to tokenize and embed log texts. Observing that the domain-specific information is in noun words representing the names of entities. Other words, e.g., verbs, are borrowed from natural languages. The meaning of these words (e.g., verbs) should be customized, which can be handled by training or customizing a language model. For nouns, the names of these entries follow standard naming conventions and use predefined separators to reflect the structure. For instance, Linux uses the slash symbol as a separator for paths, while Windows uses a backslash. After splitting these names into individual words, many of them will be common English words. For common unknown words, e.g., abbreviations used in an organization, standard Linux directory names, and process names, we extend the dictionary of the language model to recognize them.

Challenge II: Another challenge is that security data is typically unbalanced. Despite the increasing number of attacks in the real world, the number of malicious activities is far less than that of benign activities. Therefore, the training data is highly unbalanced, with the majority or even all of them being benign [3]. Such unbalanced data will lead to the poor performance of a trained model. This is a typical yet open challenge in the machine learning community. A popular solution is to augment the dataset by collecting more real-world data or synthesizing data. It is challenging to obtain logs containing malicious behaviors in practice. Unlike images or data in other domains, logs are more diverse and have constrained formats and semantics. Generating diverse and high-quality security data is still an open problem in data synthesis research. As such, traditional data augmentation is impractical.

In AIRTAG, we adopt another approach, which is one-class machine learning models. These models are designed for unbalanced datasets that contain only one label, or most of the dataset belongs to one class. To be more specific, we use a one-class support vector machine (OC-SVM) which shows the best practical results among all alternative methods (§3.4). It works by learning a model only on benign data and using a function to measure the distance of individual samples to the learned pattern. If the distance is considerable, the model classifies the sample as suspicious.

Results: We evaluated AIRTAG with 19 scenarios containing both single-host attacks and multi-host attacks. Our results show that AIRTAG is 2.5x faster than the state-of-the-art method, ATLAS. Its true-positive rate is 9.0% higher, and its false-positive rate is 16.5% lower than ATLAS, demonstrating that AIRTAG is more effective compared with ATLAS. Our code can be found at `https://github.com/dhl123/Airtag-2023`.

2 Design of AIRTAG

2.1 Overview

We show the overview of AIRTAG in Figure 2. AIRTAG consists of three components, i.e., data preprocessing (Step A), training (Step B), and attack investigation (Step C). First, AIRTAG preprocesses logs from different sources, e.g., Firefox, DNS, and security events, by sorting the log entries and merging them into a single log file (Step A). Then, AIRTAG tokenizes log files, performs unsupervised learning based on BERT to generate a pre-trained model, and then fine-tunes a downstream classifier (Step B). Particularly, we design a novel tokenizer that leverages log files' domain-specific syntaxes and semantics. We use a one-class support vector machine (OC-SVM) as our downstream classifier to overcome the unbalanced data problem. During attack investigation, AIRTAG first tokenizes and embeds the query and then extracts attack-related events and marks them as `malicious`. We also leverage existing heuristics to filter out false positives caused by using one-class classification. Lastly, AIRTAG can reconstruct the attack story by generating the causal graph or reporting all suspicious events.

Scope and assumptions. Consistent with existing attack investigation approaches [3, 15, 21, 24, 39, 42], we assume the integrity of the log. Log collection and strange systems are well-protected by design and operational protocols in the real

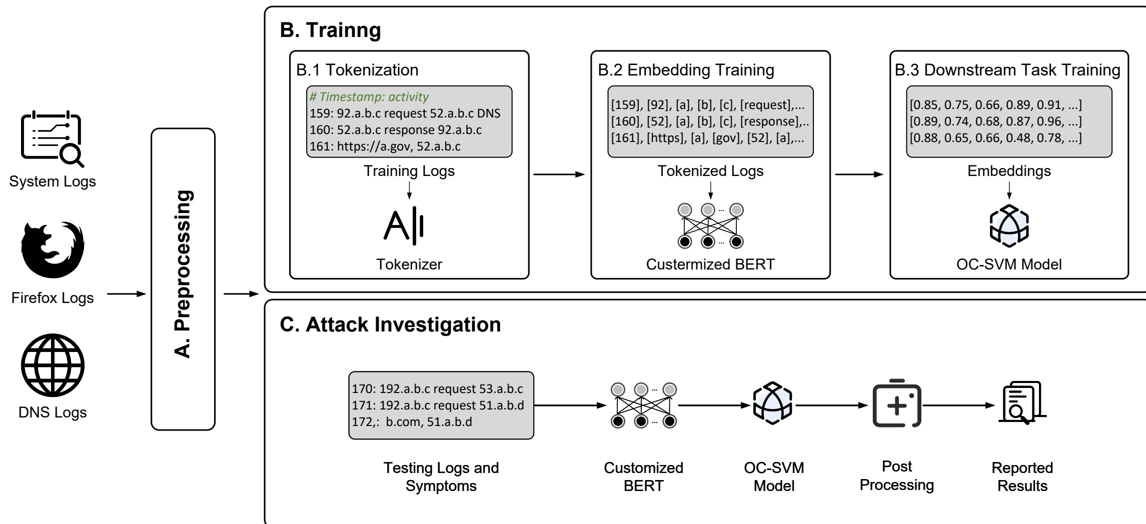


Figure 2: Overview of AIRTAG. AIRTAG consists of three components. AIRTAG preprocesses logs by sorting entries with their timestamps. Then, AIRTAG fine-tunes an unsupervised model to provide embeddings and trains an OC-SVM model for downstream tasks. Finally, AIRTAG discloses all log entries related to the attack and post processes these entries to report final results. Results can be entries or a graph.

world. Attacks that corrupt or tamper logs and their defenses are beyond the scope of this paper.

2.2 Preprocessing

The preprocessing in AIRTAG is a standard procedure that merges logs from different sources to enable correlations of logs. We merge logs with timestamps and their relations (similar to ATLAS). Logs representing the same behavior from different sources (e.g., DNS, firefox, and syslog) are clustered and merged to help capture causalities inside same behaviors. Figure 1-B.1 shows an example of merged logs. At time 159, Firefox requested resolving the domain name `https://a.gov` and sent an HTTP request to address `52.a.b.c` at time 161. In between, the DNS server resolved the domain name `https://a.gov` and got the corresponding IP address `52.a.b.c`. Notice that the source logs in Figure 1-B.1 can be in any format (e.g., both system log and DNS log formats) and do not need to be an entity-to-entity structure.

2.3 Training

Our training aims to learn a model that captures the inner relationships (i.e., causalities) among events. Our training consists of three steps: tokenization, embedding training, and downstream task training.

2.3.1 Tokenization

Tokenization is a process of separating text data into smaller units called tokens. Tokens can be words, characters, or subwords (e.g., n-gram characters). For example, the BERT model inserts a `[CLS]` token at the beginning of a given text and a `[SEP]` token between two sentences to separate them. It contains around 30,000 tokens including many special ones, `##ed`, `##ly`, and `##ing`, which can reflect its tense, POS etc. A complex word will be tokenized to its word stem and one or many special tokens. For example, the word `embeddings` corresponds to four tokens, i.e., `[em]`, `[##bed]`, `[##ding]`, and `[##s]`. There is also a special token `[UNKNOWN]` to represent all unknown words.

Using the tokenizer can reduce the vocabulary table size and also partially reflect the semantics of the words/sentences. English has more than 170,000 words, and training in such a large dictionary requires a significantly large model, which is still very challenging. Moreover, the language keeps evolving, and domain-specific words are not in the training datasets. Tokenization used in BERT is far smaller and easier to train. Also, tokens like `##ed` can be easily associated with tense during training, which makes it easier to train.

Logs are similar to natural languages because they are designed for humans to read and understand, but they are also not common natural languages. If using the traditional tokenizer, we will end up with a lot of `UNKNOWN` words and miss important semantics in the log. For example, paths are common in logs, but most of them will be recognized as `UNKNOWN` using the traditional tokenizer. In fact, paths are sequences of folder/directory names separated by pre-defined separators. When downloading files, Firefox creates a temporary file, usually named `Firefox.xxx` where `xxx` is a hash value. This is a typical design for other programs like Chrome as well. Viewing them as a whole token gives us another `UNKNOWN` word, but splitting them with domain knowledge can help us to associate the temporary files with corresponding processes during model training.

In AIRTAG, we design domain-specific tokenizers to serve

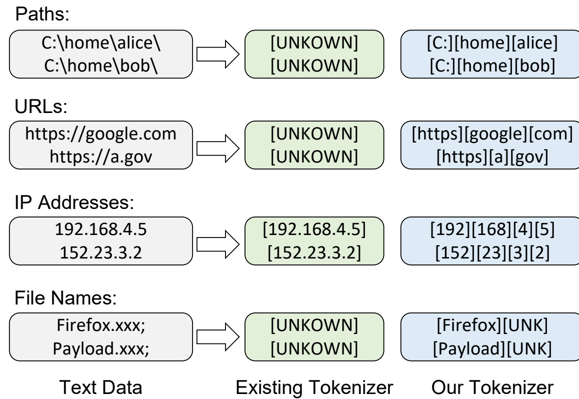


Figure 3: Comparison of Existing Tokenizer and Our Solution. Existing tokenizers mark most file paths and other log-specific items as `UNKOWN`. AIRTAG, which utilizes unique features of logs, splits these items into several fine-grained words and preserves the semantics.

a similar role to the tokenizer in traditional NLP tasks. Our tokenizers split the text by pre-defined characters such as "`.`" and "`//`". Unlike traditional tokenizers that split all text at once, we are more fine-grained. We first scan values of each field in each log entry, such as finding the URLs in the log entry, and then split the URLs accordingly. When there are multiple fields in a log entry, we perform split multiple times. Notice that we can still reuse existing tokenizers for most words in logs, and all domain-specific tokenizers are designed for names of logged entities that can be split into smaller units, such as paths, URLs, IP addresses, and temporary file names. The basic idea is to leverage pre-defined separators in these names to split them into individual words and then tokenize each word. Figure 3 shows a few examples. The path `C:\home\alice` is split into `[C:]`, `[home]`, and `[alice]` tokens by identifying the special slash symbol. Similarly, `C:\home\bob` will be `[C:]`, `[home]`, and `[bob]`, which hints the directory hierarchy. Notice that for words `home`, `alice`, and `bob`, we directly reuse the BERT tokenizer. URLs can be viewed as a combination of network protocols, domain names, ports, and a path. For instance, the `https://a.gov` uses `https`, `a`, and `gov` as its domain name. For IP addresses, we use the dot symbol to separate them directly, and IPs in the same subnets can be learned by the model. For temporary files like `Firefox.xxx`, AIRTAG also splits them into `Firefox` and `xxx`, which makes it easier to associate the file to corresponding processes. To handle different types of log formats, we leverage the log parsing framework, LogStash [1].

2.3.2 Embedding Training

Instead of training from scratch, AIRTAG leverages the BERT model to perform unsupervised learning. BERT is a bidirectional Transformer designed model for NLP understanding tasks. To customize it for our scenarios, we use it as the initialized model and retrain it using our unlabeled log data.

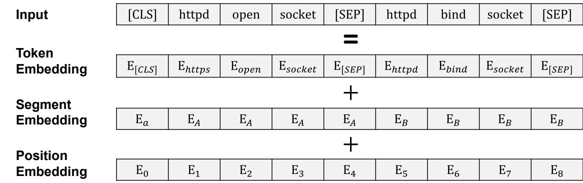


Figure 4: Embeddings in BERT. The output embedding is a composition of the token, segment and position embedding.

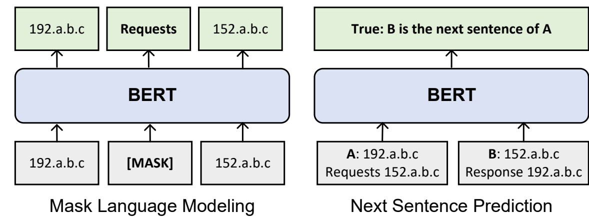
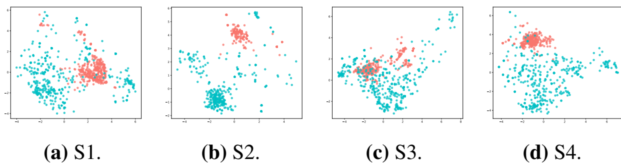


Figure 5: Different Training Tasks in BERT. The goal of masked language modeling task is to predict a masked word based on the contextual information of the word. Next sentence prediction aims to pair two sentences and determine whether the former sentence is the antecedent of the latter sentence.

Embedding. We basically followed the design of BERT and designed three embeddings. As shown in Figure 4, the final embedding of the input combines token embedding, segment embedding, and position embedding. AIRTAG reuses the BERT token and positional embedding. BERT views each sentence as a single segment, and tokens in the same sentence have the same segment embedding (also known as sentence embedding). In our scenario, there is no sentence concept. AIRTAG treats each event as a unit, and all tokens belonging to the same event have the same segment embedding.

Training objectives. Similar to BERT, we also leverage the masked language model (MLM) and next sentence prediction (NSP) (in AIRTAG, it is the next segment/event prediction) to train our model. The basic idea of MLM is to replace a token or a segment in the log with a special placeholder token `[MASK]`, and during training, we use the context information (e.g., surrounding tokens and segments) to predict the concrete values of the `[MASK]`. By doing so, the model learns the encoding of each word and segment by learning from its contexts, reflecting the semantics. For the log examples shown in Figure 5, we mark `requests` in `192.a.b.c requests 152.a.b.c` as `[MASK]`. Then, we try to use `192.a.b.c [MASK] 152.a.b.c` to predict the concrete content of `[MASK]`. Such mask language prediction captures the relations between `192.a.b.c` and `152.a.b.c`. On the other hand, NSP focuses on predicting whether the second sentence in a given sentence pair is a follow-up of the first sentence. As shown in Figure 5, BERT takes paired sentences `A: 192.a.b.c requests 152.a.b.c` and `B: 152.a.b.c responses 192.a.b.c` as the input and predict `True`, meaning that B is the next sentence of A. In practice,



(a) S1. (b) S2. (c) S3. (d) S4.

Figure 6: The red and blue color indicate the embedding vector of malicious and benign entries, respectively. We only show 500 malicious and 500 benign samples to make the figure readable.

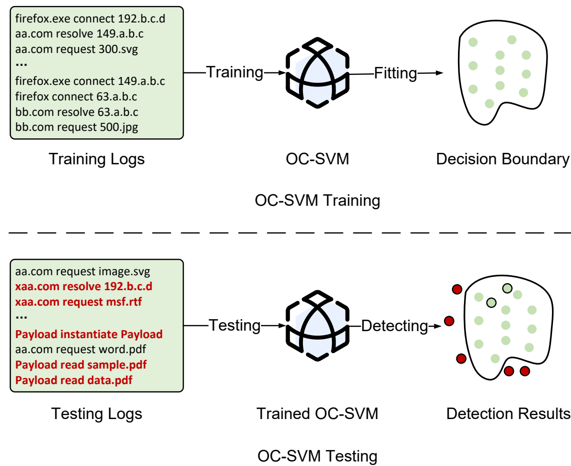


Figure 7: Downstream Task Training and Testing in AIRTAG.

the next sentence can be a combination of several log entries/sentences within a window (rather than a single temporally consecutive log entry). Thus, BERT can capture complex relationships between multiple contextual entries.

2.3.3 Downstream Task Training

After unsupervised learning on a large set of data, the trained model learns the inner relationships among tokens, which can be used for different downstream tasks. We use PCA to reduce the dimension of embedding vectors and visualize them in Figure 6. The blue dots show the embedding vectors of benign entities and the red ones indicate malicious entries. As the results show, embedding vectors of benign and malicious entries are different, making it possible to classify them. We fine-tune it for specific downstream tasks to capture such patterns. A typical workflow is to plug a model (e.g., a linear classifier) with pre-trained models and leverage a small set of labeled data to perform supervised learning to train the model. For example, BERT can learn the relationships of `good` and `bad`, and fine-tuning with a classifier will train a model that searches for such words for sentiment analysis. By fine-tuning the model on task-specific datasets, the model can achieve better results on downstream tasks.

During attack investigation, AIRTAG needs to determine if a given logged event is malicious (should include in the final report) or benign (can exclude from the final report), which is our downstream task. As mentioned in §1.1, unlike AI tasks, there are not enough high-quality training datasets for log analysis, and synthesizing such datasets is challenging. The dataset we can use to train downstream tasks is unbalanced, containing only (or mostly) benign patterns. Thus, it will be hard for us to train a classifier to differentiate different behavior patterns. To alleviate this problem, we use one-class classification techniques. Specifically, we use a one-class support vector machine (OC-SVM), which achieves the best results among all one-class classification techniques. The OC-SVM classifier tries to learn benign patterns and classify samples far from the learned benign pattern as suspicious. As illustrated in Figure 7, OC-SVM learns benign behaviors from unlabeled training logs (i.e., the benign ones) and tries to find a decision boundary that fits training data. When detecting outliers, OC-SVM classifies inputs that are not within the decision boundary as malicious (indicated by the red dots). Besides training on benign data, OC-SVM can also be trained on datasets that are not completely clean, as long as most of the data is benign. Because OCCs learn patterns of the majority of activities, few attack activities are naturally filtered out when models converge.

2.4 Attack Investigation

The goal of the attack investigation is to recover the whole attack story based on the given audit logs. Just like using BERT for predictions, we first tokenize the log and symptom event and then use the pre-trained model and OC-SVM to find all suspicious events, which can reconstruct the whole attack story. Our post-processing leverages commonly used heuristics to filter out the results (e.g., the frequency of events [39]). Then, AIRTAG constructs the causal graph using the small-sized reported suspicious events. The list of all post-processing rules are included in §A.1.

As shown in Figure 8, AIRTAG generates causal graphs from the detected events just like ATLAS. Specifically, AIRTAG generates an initial attack graph on log entries that are classified as malicious by OC-SVMs. Since AIRTAG only reports individual events based on log texts, it is possible that the aforementioned graph construction process generates disjoint graph components (Figure 8-A). As such, AIRTAG correlates those disconnected graph components and returns the complete causal graph as follows.

For each individual graph, AIRTAG expands it by finding nodes that are adjacent to other graphs from the original events in log texts (i.e., the orange nodes in Figure 8-B). If two components share a common node (red nodes), AIRTAG merges the two disconnected graph components by adding the node and its corresponding edges (Figure 8-C). AIRTAG repeats this process until all the disjoint graph components are connected. In this process, AIRTAG only tries to find the first matched event for any two disjoint graph components. This process only requires creating the initial graph once and

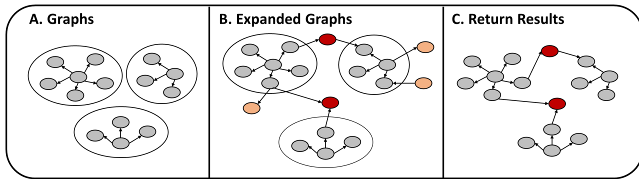


Figure 8: The process of graph reconstruction in AIRTAG. Red nodes are shared nodes and orange ones are expanded nodes.

connecting each two disjoint components once without rebuilding the graph repeatedly. In practice, to reduce the time complexity of the search, we can limit the search depth to a small number (e.g., 1) and can still recover the attack story (§3.5). The reconstruction algorithm is a greedy method that does not guarantee the correctness of selected paths, it only chooses the most probable ones.

In addition, similar to SPADE [17], we explicitly check network operation pairs (i.e., request and response) if only one of them is included in the original causal graph. In this way, AIRTAG recovers the attack story with more contexts which are helpful for attack investigation.

3 Evaluation

We built a prototype of AIRTAG and evaluated it on 19 attack scenarios. This section first describes our experiment setup, including the development framework, running machine hardware and software configurations, and datasets (§3.1). We then evaluate the effectiveness of AIRTAG on attack investigation by comparing the true positive rate (TPR) and the false positive rate (FPR) of AIRTAG with those of ATLAS (§3.2). To evaluate the efficiency of AIRTAG, we measure the time costs of AIRTAG and compare AIRTAG with existing work (§3.3). In addition, we conduct an ablation study to evaluate more attack scenarios, methods, the impacts of configurable parameters, and other factors that may affect the performance of AIRTAG (§3.4). Finally, we show four causal graph cases generated by AIRTAG (§3.5).

3.1 Experiment Setup

The prototype of AIRTAG is implemented in Python 3.6.3 with ThunderSVM [66]. All experiments are conducted on a Ubuntu 18.04 machine equipped with a GeForce RTX 6000 GPU, 64 2.30GHz CPUs, and 376 GB of main memory.

Datasets. We use public datasets provided by ATLAS (S1 to S4 and M1 to M6), datasets collected by ourselves (U1 to U3 and U-Step1 to U-Step3), and datasets used in DEPIMPACT [15] (dataleak, vpnfilter, and shellshock), which cover 19 real attack scenarios and contain 25 log file sets (logs of each ATLAS dataset and our generated dataset includes system, DNS and firefox logs, and logs in other datasets are sysdig logs) collected from different victim hosts. We show the overview of the datasets in Table 1, including the dataset names, attack types, attack targets, the number of known attack clues (e.g., attacker's IP., etc.), raw log sizes, the ratio of nodes involved in both benign and malicious activities over all nodes in malicious activities, and the ratio of malicious activities over all activities in each column, respectively.

More specifically, datasets S1 to S4 include attacks performed on a single victim host. M1 to M6 collect attacks performed on multiple victim hosts. In the first host, the attacker replaced a benign web page in the victim system with a malicious one. In the second host which connected to the first host, the victim accessed the malicious web page and got compromised. Each attack performed on multiple hosts consists of two log files which are collected from the two victim hosts, respectively. We use the dataset name plus the host id to refer to each log file. For example, M6h1 refers to logs collected on the first victim, and M6h2 refers to logs collected on the second host.

Unlike datasets provided by ATLAS that only cover successed attacks, our own datasets are collected from failed attacks (U1 to U3 and U-Step1 to U-Step3). We collect the datasets similarly to ATLAS datasets (i.e., same environment, attack type, and similar benign behavior workload). Attack in each dataset follows the same attack steps as ATLAS attack with the same attack type name in the Table 1. The only difference is that we forced certain attack steps to fail. Specifically, U1 to U3 datasets are generated during an entire process of different failed attacks. The attacks exploited the vulnerability of victim systems, injected payload, and used the payload to read files. However, the attacker cannot upload the leaked data to their hosts due to network issues. We use U1 to U3 datasets to evaluate the overall effectiveness of AIRTAG on failed attacks. U-Step1 to U-Step3 are logs collected after each attack step (exploit the vulnerability, upload payload, and payload reads sensitive files). We use them for fine-grained analysis of AIRTAG on failed attacks. For the benign workload, we include browser behaviors, system behaviors, and application behaviors. To simulate normal browser behaviors, we control the machine to visit different websites such as different Wikipedia pages, search with prompts by different search engines (e.g., Google and Baidu), download different files from various websites, etc. Our system behaviors include normal behavior of system processes, and our application behaviors contain activities of widely used applications such as Microsoft applications and notebooks, etc.

Dataleak, vpnfilter, and shellshock datasets have more complicated benign behavior workloads as demonstrated by the low ratio of malicious behaviors in Table 1. We include these datasets to analyze the sensitivity of AIRTAG towards more complicated benign workloads. Specifically, we choose the datasets used in the original paper [15], which cover data leak, VPN filter, and shellshock attacks. Details of these attacks can be found in Section 5.1.2 of the original paper.

Ground truth labeling. Although AIRTAG does not require

Table 1: Overview of the 19 Attack Scenarios used by AIRTAG. BS, MS and Win are shorts for browser, MicroSoft office and Windows.

Dataset	Attack Type	Target	#Clue	Size(MB)	%Overlap	%Malicious
S1	Web compromise	BS	3	382	78.68%	6.46%
S2	Malvertising	BS	3	1015	84.00%	4.30%
S3	Spam campaign	MS	6	522	76.96%	12.12%
S4	Pony campaign	MS	5	449	84.65%	16.17%
M1	Web compromise	BS	6	711, 102	83.85%	4.06%
M2	Phishing	BS	5	671, 112	85.33%	13.68%
M3	Malvertising	BS	4	336, 138	86.12%	11.60%
M4	Monero miner	Win	6	533, 91	83.69%	3.80%
M5	Pony campaign	MS	5	726, 113	87.36%	5.33%
M6	Spam campaign	MS	6	551, 142	79.13%	4.18%
Dataleak	Data leakage	Shell	7	38	85.71%	0.07%
Vpnfilter	Malware	IoT	4	222	100.00%	0.001%
Shellshock	Bashdoor	Shell	9	84	87.50%	0.003%
U1	Pony campaign	MS	4	236	54.35%	1.20%
U2	Malvertising	BS	2	139	46.15%	2.37%
U3	Phishing	BS	3	251	48.28%	1.21%
U-Step1	Phishing	BS	1	164	11.76%	0.42%
U-Step2	Phishing	BS	2	224	23.81%	0.35%
U-Step3	Phishing	BS	3	303	46.43%	0.91%

labeling data by design, we obtain ground truth labels for evaluation purposes. Following ATLAS, we mark the given malicious entities, their neighborhood, and related activities in the graph as malicious. Since ATLAS labels data at the entity level, we further attribute the attack entities and their activities to corresponding log entries. All associated log entries are labeled malicious, and the rest are benign. Finally, we manually cross-checked the labeled log entries to ensure they included the entire attack story. For dataleak, vpnfilter, and shellshock datasets, we follow their original labels, which mark multiple annotated critical edges (we attribute the corresponding entries) as malicious.

3.2 Effectiveness of Attack Investigation

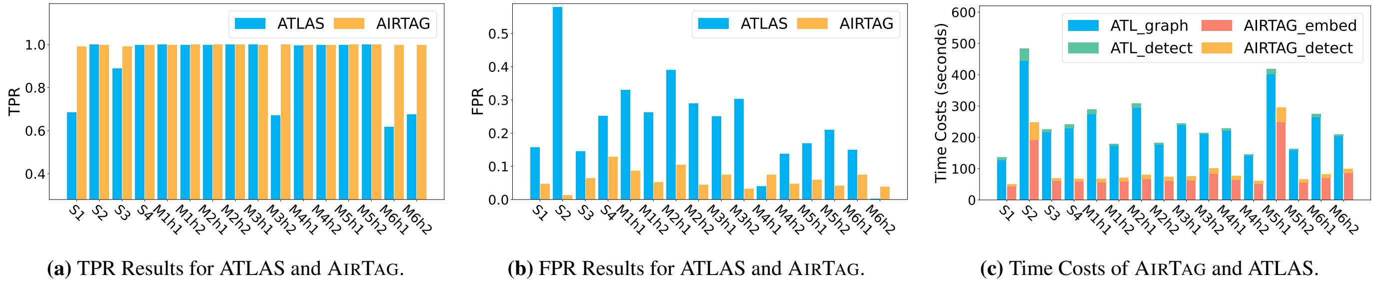
To demonstrate the effectiveness of AIRTAG on attack investigation, we measure the true positive rate (TPR) and the false positive rate (FPR) of AIRTAG, and compare the results with those of ATLAS in detecting suspicious log entries. The TPR is defined as the number of detected attack entries over the total number of all attack entries. The FPR is the number of benign entries that are misclassified as malicious over the number of all benign log entries. Each set of reported TPR and FPR are from the same model. If not specified, we reuse the original parameter settings for methods with their original implementation, which we believe is already optimal. For example, we use its own datasets' original settings in ATLAS. For others without the original implementation, we report the best empirical results obtained by tuning hyperparameters on a small dataset, following common practice in machine learning hyperparameter tuning. We show the results in Figure 9. When investigating an attack conducted on a single victim, we train AIRTAG and ATLAS on other datasets of its kind. For example, the results of S1 are obtained by training AIRTAG on benign entries in datasets S2, S3 and S4. For logs collected on multiple hosts, we test two attacks each time (M1 and M2, M3 and M4, M5 and M6), using the logs collected from other multi-host attacks as the training data. For example, the testing results on M1 and M2 are generated by AIRTAG trained on M3 to M6. Here, we would also like to clarify that our tested data is different from the training data, where the test data includes new attacks and activities. Our goal is to test whether the models trained on the training data can generalize to the *unseen* test dataset without retraining. Also, different from AIRTAG that are completely automatic, ATLAS relies on manually specifying a clue entity as the start point for attack investigation. In this setting, using different clues as the starting point derives different results. Therefore, we report the average results of using different clues and include an analysis for different start points in §3.4.

From the results, we observe that AIRTAG achieves better TPR and FPR compared to ATLAS. The average TPR and FPR of AIRTAG are 99.8% and 6.2%, which are 9.0% higher and 16.5% lower than those of ATLAS, respectively. By leveraging large-scale embedding models, OC-SVM, and filters, AIRTAG can achieve better performance than advanced learning-based methods over the causal graph, indicating that log-level attack investigation is feasible and promising.

Although the FPR of AIRTAG is slightly higher than that of ATLAS in M6h2, the difference is rather small (3.6%), and the TPR of AIRTAG is significantly higher than that of ATLAS (32.3% higher). The main reason behind such cases is that ATLAS cannot precisely differentiate attack behaviors from benign ones. ATLAS conservatively classifies most entries as benign, leading to low positive rates (FPR as well as TPR). For a unique case M4h1, AIRTAG and ATLAS achieve similar results (the difference between TPR is 0.2% and FPR is 3.4%). The potential reason is that attack patterns inside M4h1 are more easily to be identified. Therefore, both AIRTAG and ATLAS can achieve good results.

Effectiveness on failed attacks. To further understand the effectiveness of AIRTAG and ATLAS against failed attacks, we run AIRTAG to investigate three failed attacks and each step of a failed attack. Specifically, we train AIRTAG on two unsuccessful attack datasets (two datasets from U1 to U3) and use the other one as testing data. To evaluate the performance against each attack step, we use the model trained on U1 and U2 datasets to investigate attacks inside U-Step1, U-Step2 and U-Step3 datasets. The results are shown in Figure 10. Figure 10(a) and Figure 10(b) show the overall performance of AIRTAG on U1 to U3 datasets. Figure 10(c) and Figure 10(d) demonstrate the results on each attack step.

The results show that while both AIRTAG and ATLAS can be used to investigate failed attacks, AIRTAG performs better. The average TPR and FPR results of AIRTAG are 99.23% and 14.75%, which are 7.55% higher and 14.08% lower than those of ATLAS. On the one hand, AIRTAG and ATLAS exploit both global and local information, so they can still infer malicious entities based on local information, despite the incomplete attack chain. On the other hand, AIRTAG achieves

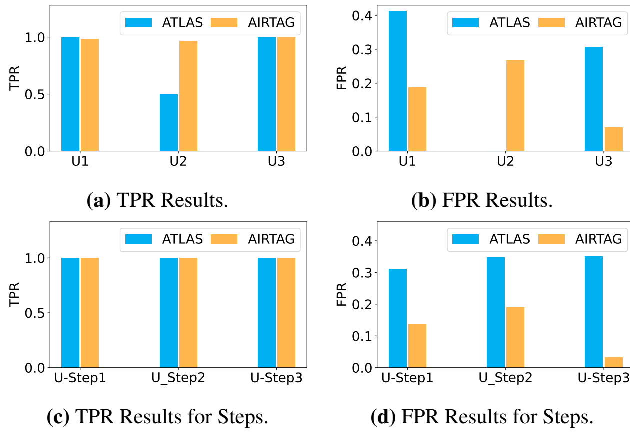


(a) TPR Results for ATLAS and AIRTAG.

(b) FPR Results for ATLAS and AIRTAG.

(c) Time Costs of AIRTAG and ATLAS.

Figure 9: Evaluation Results for Attack Investigation. ATL is short for ATLAS.



(a) TPR Results.

(b) FPR Results.

(c) TPR Results for Steps.

(d) FPR Results for Steps.

Figure 10: Results on Failed Attacks.

better performance than ATLAS by design, which is also confirmed by results in §3.2.

3.3 Time Costs for Attack Investigation

To measure the efficiency of AIRTAG, we evaluate the time costs of attack investigation for both AIRTAG and ATLAS. During the attack investigation, ATLAS converts the logs into a causal graph and then does classification. Therefore, we show both the total time costs of ATLAS and the time costs of its causal graph-related operations. AIRTAG converts logs to embedding vectors and then detect attacks. Thus, we show the total time costs, the costs of embedding and detection for AIRTAG. Figure 9(c) summarizes the results.

From the results, we observe that the time costs of AIRTAG are lower than the time costs of ATLAS in all cases. Specifically, the average cost of AIRTAG is 39.96% of ATLAS. The results show that AIRTAG, which directly investigates attacks at the log level and omits causal graph-related operations, is more efficient than ATLAS. Directly investigating attacks at the log level is beneficial.

When analyzing the time cost composition of ATLAS, we find that the main time costs of ATLAS come from the causal graph-related operations, including reading and operating on causal graphs. The average time cost of causal graph-related operations is 3.94 minutes, which is close to the the average time cost of 4.10 minutes for the whole attack investigation process in ATLAS. As shown by the results, the high time cost of causal graph-related operations is the bottleneck of existing causal graph-based methods. When comparing the total time costs of AIRTAG with the time costs of causal graph-related operations in ATLAS, the total time cost of AIRTAG is only 41.54% of graph-related operations in ATLAS. AIRTAG, which directly investigates attacks at the log level, can solve such a bottleneck and speed up the investigation by omitting causal graph-related operations.

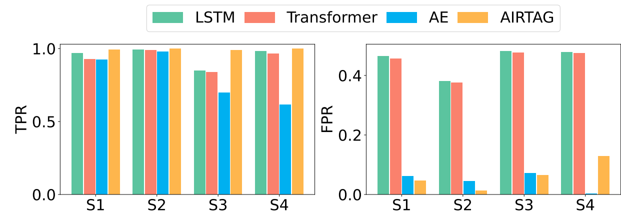


Figure 11: Comparison with Anomaly Detection.

3.4 Ablation Study

We evaluate AIRTAG under more scenarios, measure the factors that may affect AIRTAG and ATLAS performance, and discuss potential sensitivities of AIRTAG.

Comparison with anomaly detection. Although AIRTAG focuses on attack investigation tasks with very different goals than anomaly detection tasks, some anomaly detection techniques may be adaptable. To measure this, we compare AIRTAG with three widely recognized unsupervised anomaly detection methods (i.e., LSTM-based, Transformer-based, and AE-based methods) summarized in deep-loglizer [6]. Since these anomaly detection methods use abstractions that are specific to the used datasets and cannot generalize to different log formats (e.g., simplifying *blk* values and other specific file information that is not avaliable in other logs), we did our best to customize a similar abstraction that also abstracts specific file information into a general representation. Other implementation details, and settings, are the same as the original implementation. We summarize the results in Figure 11.

We observe that AIRTAG obtains better performance com-

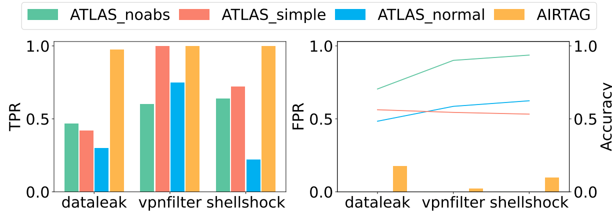


Figure 12: Results on Depimpact Datasets.

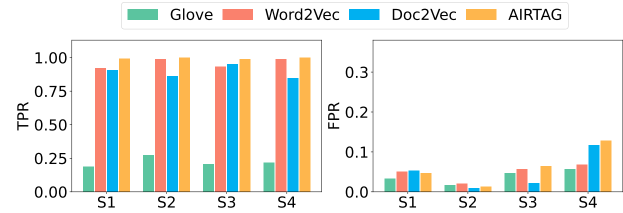


Figure 13: Results of Using Different Embedding Methods.

pared to anomaly detection methods. On average, AIRTAG has 4.75%, 6.65%, and 19.07% higher TPR compared to LSTM-based, Transformer-based, and AE-based anomaly detection methods. AIRTAG also gets 38.80% and 38.28% lower FPR compared to the LSTM-based and Transformer-based methods. Although AIRTAG achieves 1.78% higher FPR than the AE-based method (mainly caused by S4), the TPR of the AE-based method in S4 is significantly lower than that of AIRTAG in S4. The better performance of AIRTAG is understandable because AIRTAG considers more the whole attack story rather than the isolated abnormal values that the anomaly detection methods mainly consider.

Performance on more datasets. To test whether AIRTAG is sensitive to more complicated benign behaviors and attacks, we evaluate AIRTAG on the dataleak, vpnfilter, and shellshock datasets, and compare AIRTAG with ATLAS. Since ATLAS abstracts specific file and process names to general names by examining their prefixes such as converting process names containing prefix `c:/programfiles` to `programfiles_process`, which is specific to a particular computer machine and Windows logs, we tried our best to reproduce the abstractions for new datasets that contain mainly Linux logs. We also include the performance when no abstraction is applied or when all filenames and processes are simply abstracted to `file` and `process`. We show the results in Figure 12. ATLAS_noabs, ATLAS_simple and ATLAS_normal show the results without abstraction, with simple abstraction and with our reproduced fine-grained abstraction. The lines show the training accuracy of ATLAS.

We found that AIRTAG still outperforms ATLAS because the TPR of AIRTAG is significantly higher than that of ATLAS, and the FPR of AIRTAG is also low. Specifically, the average TPR of AIRTAG is 99.16%, while the TPR of ATLAS is only 56.89%. The possible reason for poor performance of ATLAS is that they assume attacks have similar behavior patterns and try to learn them. ATLAS may fail when the attacker changes the strategy and uses very different attack activities (e.g., attacks in the dataleak, vpnfilter, and shellshock datasets). Unlike ATLAS, AIRTAG learns most benign behaviors and thus has better generalization to attacks with different strategies.

Different embedding methods. To test the impacts of using different embedding methods, we replace BERT embedding in AIRTAG with a statistical-based embedding method (Glove [50]) and two learning-based methods (Word2Vec [41] and Doc2Vec [32]). The results are shown in Figure 13.

As shown in the figure, AIRTAG achieves the best TPR results on all datasets. Specifically, the average TPR of AIRTAG is 99.5%. It is 77.4%, 3.7%, and 10.3% higher than that of Glove, Word2Vec, and Doc2Vec, respectively. Results show the benefits of AIRTAG. We also notice that Glove achieves the worst result. This is reasonable because Glove only uses statistical information and ignores the causalities inside logs.

The FPR of AIRTAG is also comparable with other methods. The average FPR differencees between AIRTAG and other methods are 2.5%, 1.4%, and 1.3%, respectively. Although in some cases (e.g., S3 and S4), the FPR of AIRTAG is slightly higher than those of Word2Vec and Doc2Vec (4.2% higher than Doc2Vec in S3 and 6.0% higher than Word2Vec in S4), AIRTAG achieves significantly higher TPR. We consider TPR as more important than FPR because missing malicious events can lead to worse results than introducing false alarms. Since the TPR of AIRTAG is the highest among all methods and the FPR is also comparable, we consider BERT more suitable for attack investigation.

Different classifiers for downstream tasks. As mentioned earlier, AIRTAG implements an OC-SVM with RBF kernel function as the downstream task classifier. To justify this design choice, we evaluate the FPR and TPR of using different OCCs in AIRTAG. Specifically, we evaluate OC-SVM models with different kernel functions (i.e., linear, sigmoid and RBF functions), as well as a deep neural network-based OCC model (i.e., Ocgan [51]). Although there are other deep neural network-based OCCs, they are implemented for image processing [51, 56] which are not suitable for our task. We choose Ocgan because it is applicable for embedding vectors after our adaption. To ensure the fairness of the comparison, we use the same parameters (e.g., the nu and gamma values) for different OC-SVMs. For OCGAN, we tried our best effort tuning the parameters. The training epoch of OCGAN is 8 and the thresholds for each dataset are 0.75, 0.67, 0.73, and 0.8. Other parameter settings are default settings [18].We show the results of comparing different classifiers in Figure 14.

We observe that AIRTAG, which uses OC-SVM with the RBF kernel function, achieves much better results than OC-SVM models with other kernel functions. Compared with other OC-SVM models, the TPR of using RBF kernel functions is the highest. The results show the advantage of RBF

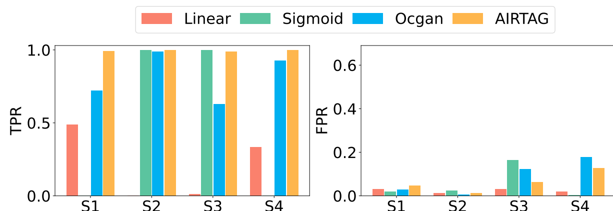


Figure 14: Results of Using Different OCCs in Downstream Tasks.

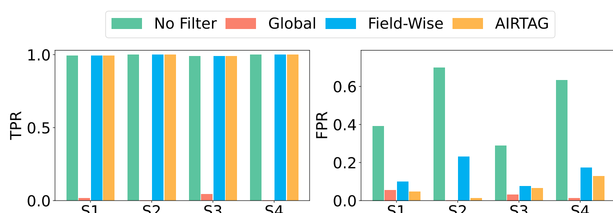


Figure 15: Results of Using Different Filters in the Post Processing.

kernel functions. Moreover, although the FPR result of using AIRTAG is not the best, other cases that achieve better FPR results have much worse TPR results. Therefore, using RBF function is still considered the best solution. Existing work [40] also favors the RBF kernel functions because they have been proven more effective than other kernel functions.

We also observe that the results of using RBF functions are more stable than using other kernel functions in our scenarios. The difference between the highest TPR and the lowest TPR is 0.9% of OC-SVMs with RBF kernel functions, which is significantly smaller than 48.8% and 100.0% for OC-SVMs with linear and sigmoid kernel functions. The results show the better generalization of OC-SVM with RBF kernel functions.

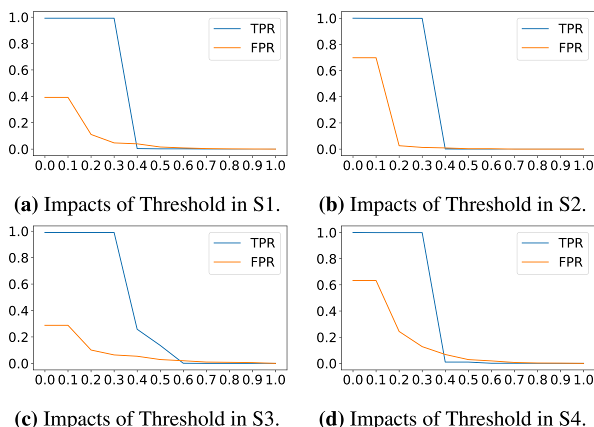
When comparing with deep neural network-based methods, the performance of using OC-SVM with RBF kernel function is also better than the results of using Ocgan. As observed, the TPR of AIRTAG is higher than Ocgan, and the FPR of AIRTAG is comparable with that of Ocgan. Specifically, the average TPR of AIRTAG is 17.9% higher than that of Ocgan. The FPR of AIRTAG is 2.0% lower than the FPR of using Ocgan. OC-SVM with RBF outperforms Ocgan because most deep neural network-based OCCs are designed for visual tasks, including Ocgan. Therefore, they may not work well for natural language-like tasks since natural language and visual data domains are quite different. Another reason is that deep neural networks can easily overfit input data, which leads to poor performance. Since using OC-SVM models with RBF kernel functions achieves the best and most stable results among all OCCs, we set OC-SVM with RBF kernel functions as our default downstream task model in AIRTAG.

Different filters. We evaluate three filters based on global frequency, field-wise frequency of test log files, and field frequency filters with tolerance bounds (default setting in AIRTAG). For global frequency filter, we count the frequencies of all words in the testing file and choose the most frequent ones (i.e., top 30% by default for ATLAS and our datasets, we evaluate such choice in Figure 16) as global frequent words. AIRTAG classifies a log entry as benign if and only if each word in this entry belongs to global frequent words. For field-wise one, we count the frequencies of words in each semantic field and pick the most frequent ones (i.e., top 30%) as frequent words of each field. AIRTAG classifies a log entry as benign if and only if words in each field belong to frequent words of this field. We observe that some randomized filenames of benign files that occur very rarely (only once) are considered malicious in the above two methods, such as the name of a counting temporary file. Therefore, we further relax the constrains in the field-wise frequency by adding a tolerance bound to allow some rare words classified benign (i.e., default filter in AIRTAG). Note that, OCCs in downstream task training can already achieve excellent TPR results because the models are trained on a large amount benign data, and can find a precise and constrained decision boundary for benign entries. Therefore, our filters only aim to filter the misclassified portion of the entries classified as positive by the OCCs. We show the results of not using filters and using different filters in Figure 15.

As inferred from the figure, the default setting of AIRTAG, which uses both field-wise frequency and tolerance bounds, achieves the best TPR and FPR. Compared with not using filters, the TPR of AIRTAG is similar but the FPR is significantly lower, showing the effectiveness of our filters. Compared with only using field-wise frequency, AIRTAG achives better performance by allowing rare words to be benign. The average FPR of AIRTAG is 6.3%, lower than that of the filed-wise method (14.3%). The results show that a tolerance bound can improve field-wise based methods and decrease the FPR because such schema reduces the false alarms caused by randomized file names of benign files.

We also observe that using global frequency achieves the worst results among all cases. The average TPR of the global frequency-based filter is only 1.5%, which is almost unacceptable. The performance of using global frequent words is poor because log entries are field-sensitive. Words that are frequent in the entire file may not necessarily be frequent in each field. Therefore, only using global frequency introduces much noise and make the performance poor.

Threshold in filters. As mentioned earlier, AIRTAG needs to set a threshold in the filters to decide which words should be selected as frequent words. Such thresholds can affect the performance of our filters and the final results of AIRTAG. Using a small threshold may result in failure to filter out false positives, but using a large threshold can incorrectly classify true positives as benign. By default, the threshold is 0.3 for ATLAS and our datasets, meaning that we consider the top 30% of words in terms of frequency as frequent. To test the effects of different thresholds on the performance of AIRTAG, we evaluate the TPR and FPR of AIRTAG under different threshold settings (i.e., from 0.0 to 1.0), and show the results



(a) Impacts of Threshold in S1.

(b) Impacts of Threshold in S2.

(c) Impacts of Threshold in S3.

(d) Impacts of Threshold in S4.

Figure 16: Impacts of thresholds in filters.

Table 2: Results of Using Different Start Points in ATLAS.

Dataset	Web Name		IP Address		Payload	
	TPR	FPR	TPR	FPR	TPR	FPR
S1	0.997	0.000	0.061	0.000	1.000	0.472
S2	1.000	0.000	1.000	0.812	1.000	0.928
S3	0.785	0.009	0.991	0.000	1.000	0.845
S4	0.997	0.000	1.000	0.447	1.000	0.812
Average	0.945	0.002	0.763	0.315	1.000	0.764

in Figure 16. Figure 16(a) to Figure 16(d) show the results on different datasets (S1 to S4).

As the results demonstrate, increasing the threshold can decrease both TPR and FPR of AIRTAG. On the one hand, increasing the threshold allows some benign log entries that are previously classified as malicious to be classified as benign, thus reducing the false positive rate. On the other hand, some positive entries may also be misclassified as benign when the threshold is too large. Therefore, the TPR also decreases when the threshold increases. As observed, the decrease in FPR is significant, and the loss of TPR is almost impervious when choosing a proper threshold. For example, when we choose 0.3 as the threshold, the loss of TPR is merely 0.1%, and the decrease of FPR is 43.98% on average. The results show that using filters is useful and they only filter false positives out. By default, we set the threshold 0.3 as the default setting. The settings may be different in different datasets. Also, we notice that for some datasets, the best thresholds are similar. This is because small thresholds do not filter any entries, while large thresholds filter all entries. Therefore, thresholds are usually naturally limited to a similar, modest number.

Different start points in ATLAS. ATLAS requires analysts providing a known attack clue as the prior knowledge for attack investigation. To test the effects of using different start points in ATLAS, we choose to use malicious webname, malicious IP address and payload as the start points of ATLAS and evaluate their TPR and FPR results. We show the results in Table 2. The last raw of the table shows the average TPR and FPR results of using different start points.

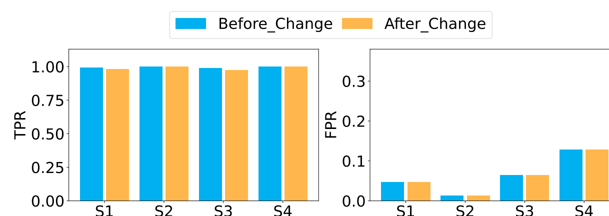


Figure 17: Sensitivity to Payload Names.

We find that ATLAS is sensitive to start points. A good attack start point can derive very good results (e.g., using malicious web names in S1, S2 and S3) but a bad start point can lead to very low TPR (6.1% when using the IP address in S1) and high FPR (the FPR of using payload in S2 is 92.8%). ATLAS is sensitive to start points because different attack clues have different impacts on the system when represented as nodes. For example, the average TPR is the highest when using payload as the start point in Table 2. This is because the payload is activity-intensive. Such node derives most edges related to attack activities (98.9%, 99.8%, 78.2%, and 99.6% attack activities are derived from payload node in different datasets). Using payload as the start point directly observes all of its relevant attack activities and achieves high TPR. When using other clues as the start point, the TPR highly depends on whether the payload can be detected or not. For example, when using the IP address in S1, ATLAS does not detect the payload and therefore, the TPR is quite low.

On the other hand, activity-intensive applications are connected to more diverse entities, making it more difficult to precisely identify the feature of relevant attack activities. Therefore, using payloads as the start points often leads to higher FPR than using other attack clues. In practice, it is already difficult to accurately find a real attack alert as attack clues among a large number of false alarms. It is even more challenging to test and find the best start point for ATLAS.

Sensitivity to specific words. To test if AIRTAG is sensitive to the meaning of specific words, we change the payload name in each testing dataset to different names and show the TPR and FPR results of investigating attacks on modified datasets. We evaluate the sensitivity to payload name because most of the malicious log entries are correlated with payload, as previously discussed. We compare the results of changing the payload names with using the results of using original data. We show the TPR and FPR results in Figure 17.

The results show that changing the payload name does not affect the results much. The average differences between TPR and FPR are 0% and 0.7%, respectively, which are impervious. Therefore, AIRTAG is robust to payload names and is not biased to specific words.

Sensitivity to *unseen* benign data. To test whether AIRTAG only learns a limited vision of benign behaviors (e.g., only learning behaviors in the training data such as visiting a specific website and cannot generalize to new websites), we count

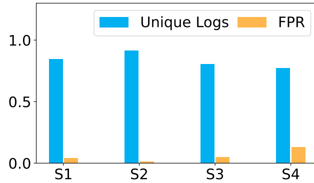


Figure 18: Sensitivity to New Benign Behavior.

the number of new files and new websites in our test data. We also calculate the ratio of *unseen* benign log entries to all benign entries in the test data. Specifically, *unseen* benign log entries are defined as benign entries in the test data that do not exist in the training data after removing time-sensitive information such as pid and timestamp. We then evaluate the sensitivity of AIRTAG to such *unseen* benign behavior (e.g., browser access to new websites) by measuring AIRTAG performance on them, and present the results in Figure 18.

We find that the new website addresses appearing in each test set are 499, 654, 2,114, and 849. For the files, these numbers are 683, 19,356, 1,105, and 693. The statistics suggest that the test dataset has included sufficient *unseen* benign objects and subjects, and the results in §3.2 demonstrate generalization over such data. According to Figure 18, we further observe that the FPR on such benign data is low. The results confirm that AIRTAG is robust to *unseen* benign data.

3.5 Case Study

In this section, we present and discuss four attack causal graph cases reported by AIRTAG from the S1, S2, S3 and S4 datasets. In our presented attack story (Figure 19), the red color indicates the attack chain and the dashed line indicates the missing entries. Edges with `Added` mean they are added by linking disconnected components when reconstructing the graph (see Section 2.4). Notice that since the original attack subgraphs are too large to be displayed, we manually simplified them by only showing the key attack steps and omits many other false positives.

Case-1: Strategic web compromise. This attack exploited CVE-2015-5122. The victim user ran a `Firefox_12` process and clicked a link which was redirected to a malicious website `Xaa.com` resolved by `192.b.c.d`. The attacker exploited CVE-2015-5122 by this link, compromised Firefox plugins and wrote a payload program `Payload.exe` to the victim computer. `Payload.exe` was executed and the derived process `Payload_1` scanned files in the victim system, established a connection to attacker, and uploaded all pdf files.

Case-2: Malvertising dominate. The whole attack process is similar to the first one. The difference is that the second case exploited vulnerability CVE-2015-3105 and compromised different browser plugins.

Case-3: Spam campaign. The attack exploited the vulnerability CVE-2017-11882. In this case, a user opened a malicious email, which has a link to the malicious website `Xaa.com` resolved by `192.b.c.d`. The user requested and downloaded a malicious file `msf.rtf`, read it with `Winword_18`. Then, `msf.rtf` established `msf_1` which writes a payload program `Payload.exe` and replaces the benign website page in the victim host with a malicious one `index.html`. The attacker executed `Payload.exe`, initialized `Payload_1`, scanned pdf files and received the pdf files.

Case-4: Pony campaign. The attacker exploited a Microsoft vulnerability CVE-2017-0199. The user opened a malicious email with a malicious word file attachment `msf.doc`. Then, the user used the vulnerable Microsoft Word program to download and open the attached Word file, and open a `Powershell_10` process. The attacker uses `Powershell_10` to upload malicious payloads and web pages. Other steps are similar to the third case.

Our results showed that AIRTAG successfully recovers the attack story. Besides, compared with ATLAS, false positives caused in AIRTAG have been significantly reduced.

4 Discussion

4.1 Limitation of AIRTAG

AIRTAG is constrained by the availability of training data because AIRTAG need to learn common benign patterns from a large amount of system behavior data (most of them should be benign). In our scenario, obtaining training data is usually feasible since a machine can generate several gigabytes of log data per day. Even if an attack was injected, most of the logs remain benign because the attacker needs keep stealthy to avoid being easily observed.

AIRTAG also may not generalize well to other log data with domain gaps. For example, if we train AIRTAG only on system logs, AIRTAG would perform relatively poorly in application logs (e.g., Firefox logs). This is a problem for all AI tasks (e.g., a model trained to translate English does not translate German well). A simple approach to improve the performance on data with domain gaps is to train the model on mixed log data (e.g., we train AIRTAG on the ATLAS dataset, which is a mix of system logs and application logs). This approach requires formatting and pre-processing for logs from different sources. Some log parsing tools, such as LogStash, can help. Another possible solution is to use domain adaptation techniques, e.g., transfer learning.

4.2 Adaptive Attacks

Similar to existing attack investigation methods, AIRTAG may be evaded when attackers carefully design their attacks. For example, attackers can make their attacks very intense and frequent, such that most of the logs are attack-related rather than benign behaviors, forcing the model to be unable to learn benign behavior patterns. However, this is not very practical in most cases since intense attack activities can be easily

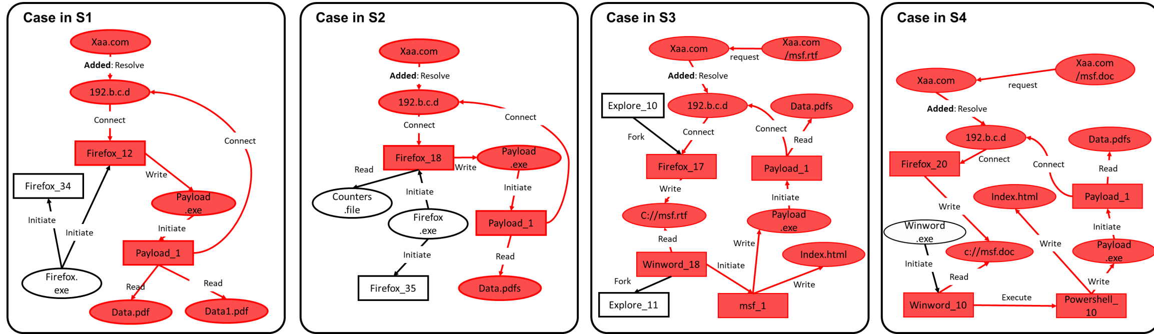


Figure 19: Investigation Cases by AIRTAG.

observed, and the defender controls the training data, who will ensure the quality of the training data.

AIRTAG depends on system log files and attacks with no trace in such files can evade it, which is a common disadvantage of system log-based systems. Such attacks include fileless and living-off-the-land attacks, where attacks only happen in memory or issue no system calls. AIRTAG also fails when the benign and malicious behaviors in log files are indistinguishable from the system log level, which inherits the limitation of machine learning-based detection.

5 Related Work

AIRTAG is related to attack detection, anomaly detection, attack investigation, word embedding and one class classifiers.

Attack detection. Existing log-level attack detection methods [44] mainly detect the existence of attacks without providing detailed analysis. Causal graph-based attack detection methods [22, 65] are popular because of their capability to capture attack behaviors. PROVDETECTOR [65] and SIGL [22] can detect the existence of attacks. At a more granular level, Unicorn [21] and RapSheet [23] can detect small compositions of attack activities or malicious processes.

Anomaly detection. There are a number of anomaly detection methods that can identify rare events or observations in a dataset that deviate from the normal behavior or expected pattern[1]. These anomalies may indicate data quality issues, fraud, or system errors. For example, Deeplog [13] uses deep learning techniques to detect abnormal behaviors in a system based on system logs, while some BERT-based methods such as LogBERT and LAnoBERT [19, 35] use advanced BERT embeddings to improve the performance. Usually, anomaly detection only detects a single or a few abnormal log entries, which is different from our target attack investigation scenario that aims to recover the whole attack story. Due to the differences between them, the design of used techniques in anomaly detection and attack investigation can be different. For example, when preprocessing logs, anomaly detection considers each single log entry, while attack investigation merges duplicated information as long as it will not affect the causalities of activities. But still, there are potential ideas in anomaly detection that we could borrow for attack investigation tasks.

Attack investigation. Different from attack detection and anomaly detection techniques (e.g., Deeplog, LogBert and LAnoBERT [13, 19, 35]) that only detect isolated outliers, attack investigation techniques aims to recover and understand the whole attack story. Existing work investigates attacks by generating and understanding causal graphs [29, 30]. Camflow [49] and other platforms [8, 11, 12, 17, 28, 62, 63, 69] provide support for collecting, storing, and searching system-level causal graphs. While the causal graph-based methods simplify the attack story, the graphs are still large and hard to understand. A series of works propose to remove unnecessary events in the graph [25, 34, 61, 67]. Beep [33] then partitions the graph into several execution units and hides detailed causalities in each unit to further compress the graph. MARSARA [68] guarantees the integrity of execution partitioning. More recently, other work also uses search-based methods [15, 39] or learning-based methods [3, 4] to prioritize the most important nodes/edges and simplify the analysis.

Embedding for text-related data. Word2vec [41] is widely used to obtain word embedding. Recently, using large language models trained on a large corpus, such as the Bert [10] and GPT models [5, 54] achieves the best results. Compared with GPT, log-based systems often use BERT because of its bi-directional architecture. We notice that there are also some embedding methods for logs such as Log2vec and Attack2vec [38, 59]. However, they are unavailable and impractical for our case. Log2vec generates graphs and performs graph embedding, which still introduces graphs and the graph embedding is different from text embedding. Attack2vec requires labeling attack events as the training data, which is labor-intensive and error-prone.

One class classifiers. Many existing works choose to use OC-SVMs [40, 47, 52] rather than other OCCs because they are explainable, easy to use and time efficient. While there are other novel alternative OCCs based on deep neural networks [2, 48, 51, 53, 57], most of them are specifically de-

[1] https://github.com/logpai/awesome-log-analysis

signed for image data rather than texts such as audit logs.

6 Conclusion

This paper proposes and builds a novel log text-based attack investigation system, AIRTAG. Specifically, we utilize several novel techniques such as unsupervised learning techniques to overcome limitations of existing causal graph-based methods. Our evaluation shows that compared to the state-of-the-art approach (ATLAS), AIRTAG achieves 9.0% fewer false positives and 16.5% more true positives. In addition, AIRTAG is 2.5x faster than ATLAS.

Acknowledgments

We thank the anonymous reviewers for their constructive comments. This material is based upon work supported by the NSF 2238847. Yuhong Nan was supported in part by the National Natural Science Foundation of China (62202510).

References

[1] Logstash. https://www.elastic.co/cn/logstash, 2022.

[2] Davide Abati, Angelo Porrello, Simone Calderara, and Rita Cucchiara. Latent space autoregression for novelty detection. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019*, 2019.

[3] Abdulellah Alsaheel, Yuhong Nan, Shiqing Ma, Le Yu, Gregory Walkup, Z. Berkay Celik, Xiangyu Zhang, and Dongyan Xu. ATLAS: A sequence-based learning approach for attack investigation. In *USENIX Security 2021*, 2021.

[4] Mathieu Barré, Ashish Gehani, and Vinod Yegneswaran. Mining data provenance to detect advanced persistent threats. In *11th International Workshop on Theory and Practice of Provenance*, 2019.

[5] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020*, 2020.

[6] Zhuangbin Chen, Jinyang Liu, Wenwei Gu, Yuxin Su, and Michael R. Lyu. Experience report: Deep learning-based system log analysis for anomaly detection. arXiv/2107.05908, 2021.

[7] Lele Cheng, Xiangzeng Zhou, Liming Zhao, Dangwei Li, Hong Shang, Yun Zheng, Pan Pan, and Yinghui Xu. Weakly supervised learning with side information for noisy labeled images. In *Computer Vision - 16th European Conference, ECCV 2020*, volume 12375 of *Lecture Notes in Computer Science*, 2020.

[8] Pubali Datta, Isaac Polinsky, Muhammad Adil Inam, Adam Bates, and William Enck. Alastor: Reconstructing the provenance of serverless intrusions.

[9] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2009*, 2009.

[10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2019.

[11] Hailun Ding, Shenao Yan, Juan Zhai, and Shiqing Ma. ELISE: A storage efficient logging system powered by redundancy reduction and representation learning. In *USENIX Security 2021*, 2021.

[12] Hailun Ding, Juan Zhai, Dong Deng, and Shiqing Ma. The case for learned provenance graph storage systems. In *USENIX Security 2023*, 2023.

[13] Min Du, Feifei Li, Guineng Zheng, and Vivek 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, CCS 2017*, 2017.

[14] En.Wikipedia.Org. 2020 united states federal government data breach. https://en.wikipedia.org/wiki/2020_United_States_federal_government_data_breach, 2021.

[15] Pengcheng Fang, Peng Gao, Changlin Liu, Erman Ayday, Kangkook Jee, Ting Wang, Yanfang Fanny Ye, Zhuotao Liu, and Xusheng Xiao. Back-propagating system dependency impact for attack investigation.

[16] Anjalie Field, Su Lin Blodgett, Zeerak Waseem, and Yulia Tsvetkov. A survey of race, racism, and anti-racism in NLP. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021*, 2021.

[17] Ashish Gehani and Dawood Tariq. SPADE: support for provenance auditing in distributed environments. In *Middleware 2012 - ACM/IFIP/USENIX 13th International Middleware Conference*, volume 7662 of *Lecture Notes in Computer Science*, 2012.

[18] Github.com. Anomaly-detection-ocgan-tensorflow. https://github.com/nuclearboy95/Anomaly-Detection-OCGAN-tensorflow, 2019.

[19] Haixuan Guo, Shuhan Yuan, and Xintao Wu. Logbert: Log anomaly detection via BERT. In *International Joint Conference on Neural Networks, IJCNN 2021*, 2021.

[20] Wenbo Guo, Dongliang Mu, Xinyu Xing, Min Du, and Dawn Song. DEEPVSA: facilitating value-set analysis with deep learning for postmortem program analysis. In *USENIX Security 2019*, 2019.

[21] Xueyuan Han, Thomas F. J.-M. Pasquier, Adam Bates, James Mickens, and Margo I. Seltzer. Unicorn: Runtime provenance-based detector for advanced persistent threats. In *27th Annual Network and Distributed System Security Symposium, NDSS 2020*, 2020.

[22] Xueyuan Han, Xiao Yu, Thomas F. J.-M. Pasquier, Ding Li, Junghwan Rhee, James W. Mickens, Margo I. Seltzer, and Haifeng Chen. SIGL: securing software installations through deep graph learning. In *USENIX Security 2021*, 2021.

[23] Wajih Ul Hassan, Adam Bates, and Daniel Marino. Tactical provenance analysis for endpoint detection and response systems. In *2020 IEEE Symposium on Security and Privacy, SP 2020*, 2020.

[24] Wajih Ul Hassan, Shengjian Guo, Ding Li, Zhengzhang Chen, Kangkook Jee, Zhichun Li, and Adam Bates. Nodoze: Combatting threat alert fatigue with automated provenance triage. In *26th Annual Network and Distributed System Security Symposium, NDSS 2019*, 2019.

[25] Md Nahid Hossain, Junao Wang, R. Sekar, and Scott D. Stoller. Dependence-preserving data compaction for scalable forensic analysis. In *USENIX Security 2018*, 2018.

[26] HP. 2020 hp bromium threat insights report q4. https://threatresearch.ext.hp.com/wp-content/uploads/2021/03/HP_Bromium_Threat_Insights_Report_Q4_2020.pdf, 2020.

[27] IBM. Cost of a data breach 2022. https://www.ibm.com/reports/data-breach, 2022.

[28] Nesrine Kaaniche, Sana Belguith, Maryline Laurent, Ashish Gehani, and Giovanni Russello. Prov-trust: Towards a trustworthy sgx-based data provenance system. In *Proceedings of the 17th International Joint Conference on e-Business and Telecommunications, ICETE 2020*, 2020.

[29] Samuel T. King and Peter M. Chen. Backtracking intrusions. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles, SOSP 2003*, 2003.

[30] Samuel T. King, Zhuoqing Morley Mao, Dominic G. Lucchetti, and Peter M. Chen. Enriching intrusion alerts through multi-host causality. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 2005*, 2005.

[31] Jack Lanchantin, Tianlu Wang, Vicente Ordonez, and Yanjun Qi. General multi-label image classification with transformers. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021*, 2021.

[32] Quoc V. Le and Tomás Mikolov. Distributed representations of sentences and documents. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014*, JMLR Workshop and Conference Proceedings, 2014.

[33] Kyu Hyung Lee, Xiangyu Zhang, and Dongyan Xu. High accuracy attack provenance via binary-based execution partition. In *20th Annual Network and Distributed System Security Symposium, NDSS 2013*, 2013.

[34] Kyu Hyung Lee, Xiangyu Zhang, and Dongyan Xu. Loggc: garbage collecting audit log. In *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS 2013*, 2013.

[35] Yukyung Lee, Jina Kim, and Pilsung Kang. Lanobert : System log anomaly detection based on BERT masked language model. Arxiv/2111.09564, 2021.

[36] Changchun Li, Ximing Li, and Jihong Ouyang. Semi-supervised text classification with balanced deep representation distributions. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021*, 2021.

[37] Xuezixiang Li, Yu Qu, and Heng Yin. Palmtree: Learning an assembly language model for instruction embedding. In *2021 ACM SIGSAC Conference on Computer and Communications Security, CCS 2021*, 2021.

[38] Fucheng Liu, Yu Wen, Dongxue Zhang, Xihe Jiang, Xinyu Xing, and Dan Meng. Log2vec: A heterogeneous graph embedding based approach for detecting cyber threats within enterprise. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019*, 2019.

[39] Yushan Liu, Mu Zhang, Ding Li, Kangkook Jee, Zhichun Li, Zhenyu Wu, Junghwan Rhee, and Prateek Mittal. Towards a timely causality analysis for enterprise security. In *25th Annual Network and Distributed System Security Symposium, NDSS 2018*, 2018.

[40] Shiqing Ma, Yingqi Liu, Guanhong Tao, Wen-Chuan Lee, and Xiangyu Zhang. NIC: detecting adversarial samples with neural network invariant checking. In *NDSS 2019*, 2019.

[41] Tomás Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *1st International Conference on Learning Representations, ICLR 2013, Workshop Track Proceedings*, 2013.

[42] Sadegh Momeni Milajerdi, Rigel Gjomemo, Birhanu Eshete, R. Sekar, and V. N. Venkatakrishnan. HOLMES: real-time APT detection through correlation of suspicious information flows. In *2019 IEEE Symposium on Security and Privacy, SP 2019*, 2019.

[43] Yisroel Mirsky, Tomer Doitshman, Yuval Elovici, and Asaf Shabtai. Kitsune: An ensemble of autoencoders for online network intrusion detection. In *25th Annual Network and Distributed System Security Symposium, NDSS 2018*, 2018.

[44] Melody Moh, Santhosh Pininti, Sindhusha Doddapaneni, and Teng-Sheng Moh. Detecting web attacks using multi-stage log analysis. In *IEEE 6th international conference on advanced computing, IACC2016*. IEEE, 2016.

[45] Kiran-Kumar Muniswamy-Reddy and David A Holland. Causality-based versioning. *ACM Transactions on Storage (TOS)*, 5, 2009.

[46] Milad Nasr, Alireza Bahramali, and Amir Houmansadr. Deepcorr: Strong flow correlation attacks on tor using deep learning. In *Proceedings of ACM SIGSAC Conference on Computer and Communications Security, CCS 2018*, 2018.

[47] Zineb Noumir, Paul Honeine, and Cedue Richard. On simple one-class classification methods. In *Proceedings of the 2012 IEEE International Symposium on Information Theory, ISIT 2012*, 2012.

[48] Poojan Oza and Vishal M. Patel. One-class convolutional neural network. *IEEE Signal Process. Lett.*, 26, 2019.

[49] Thomas Pasquier, Xueyuan Han, Mark Goldstein, Thomas Moyer, David Eyers, Margo Seltzer, and Jean Bacon. Practical whole-system provenance capture. In *Symposium on Cloud Computing, SoCC' 2017*. ACM, 2017.

[50] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014*, 2014.

[51] Pramuditha Perera, Ramesh Nallapati, and Bing Xiang. OCGAN: one-class novelty detection using gans with constrained latent representations. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019*, 2019.

[52] Pramuditha Perera, Poojan Oza, and Vishal M. Patel. One-class classification: A survey. Arxiv/2101.03064, 2021.

[53] Stanislav Pidhorskyi, Ranya Almohsen, and Gianfranco Doretto. Generative probabilistic novelty detection with adversarial autoencoders. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems, NeurIPS 2018*, 2018.

[54] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

[55] Vera Rimmer, Davy Preuveneers, Marc Juárez, Tom van Goethem, and Wouter Joosen. Automated website fingerprinting through deep learning. In *25th Annual Network and Distributed System Security Symposium, NDSS 2018*, 2018.

[56] Lukas Ruff, Nico Görnitz, Lucas Deecke, Shoaib Ahmed Siddiqui, Robert A. Vandermeulen, Alexander Binder, Emmanuel Müller, and Marius Kloft. Deep one-class classification. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018*, volume 80 of *Proceedings of Machine Learning Research*, 2018.

[57] Mohammad Sabokrou, Mohammad Khalooei, Mahmood Fathy, and Ehsan Adeli. Adversarially learned one-class classifier for novelty detection. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, 2018*, 2018.

[58] Karishma Sharma, Pinar Donmez, Enming Luo, Yan Liu, and I. Zeki Yalniz. Noiserank: Unsupervised label noise reduction with dependence models. In *Computer Vision - 16th European Conference, ECCV 2020*, volume 12372 of *Lecture Notes in Computer Science*, 2020.

[59] Yun Shen and Gianluca Stringhini. ATTACK2VEC: leveraging temporal word embeddings to understand the evolution of cyberattacks. In *USENIX Security 2019*, 2019.

[60] Eui Chul Richard Shin, Dawn Song, and Reza Moazzezi. Recognizing functions in binaries with neural networks. In *USENIX Security 2015*, 2015.

[61] Yutao Tang, Ding Li, Zhichun Li, Mu Zhang, Kangkook Jee, Xusheng Xiao, Zhenyu Wu, Junghwan Rhee, Fengyuan Xu, and Qun Li. Nodemerge: Template based efficient data reduction for big-data causality analysis. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018*, 2018.

[62] Benjamin E. Ujcich, Adam Bates, and William H. Sanders. Provenance for intent-based networking. In *6th IEEE Conference on Network Softwarization, NetSoft 2020*, 2020.

[63] Benjamin E. Ujcich, Samuel Jero, Richard Skowyra, Adam Bates, William H. Sanders, and Hamed Okhravi. Causal analysis for software-defined networking attacks. In *USENIX Security 2021*, 2021.

[64] Andreas Veit, Neil Alldrin, Gal Chechik, Ivan Krasin, Abhinav Gupta, and Serge J. Belongie. Learning from noisy large-scale datasets with minimal supervision. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, 2017.

[65] Qi Wang, Wajih Ul Hassan, Ding Li, Kangkook Jee, Xiao Yu, Kexuan Zou, Junghwan Rhee, Zhengzhang Chen, Wei Cheng, Carl A. Gunter, and Haifeng Chen. You are what you do: Hunting stealthy malware via data provenance analysis. In *27th Annual Network and Distributed System Security Symposium, NDSS*, 2020.

[66] Zeyi Wen, Jiashuai Shi, Qinbin Li, Bingsheng He, and Jian Chen. Thundersvm: A fast SVM library on gpus and cpus. *J. Mach. Learn. Res.*, 19, 2018.

[67] Zhang Xu, Zhenyu Wu, Zhichun Li, Kangkook Jee, Junghwan Rhee, Xusheng Xiao, Fengyuan Xu, Haining Wang, and Guofei Jiang. High fidelity data reduction for big data security dependency analyses. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016.

[68] Carter Yagemann, Mohammad A. Noureddine, Wajih Ul Hassan, Simon Chung, Adam Bates, and Wenke Lee. Validating the integrity of audit logs against execution repartitioning attacks. In *2021 ACM SIGSAC Conference on Computer and Communications Security, CCS 2021*, 2021.

[69] Runqing Yang, Shiqing Ma, Haitao Xu, Xiangyu Zhang, and Yan Chen. Uiscope: Accurate, instrumentation-free, and visible attack investigation for GUI applications. In *27th Annual Network and Distributed System Security Symposium, NDSS 2020*, 2020.

[70] Fei Zuo, Xiaopeng Li, Patrick Young, Lannan Luo, Qiang Zeng, and Zhexin Zhang. Neural machine translation inspired binary code similarity comparison beyond function pairs. In *26th Annual Network and Distributed System Security Symposium, NDSS 2019*, 2019.

Table 3: Post-processing Rules

Rule	Base	Target
F1	Global Word Frequency	Filter FPs
F2	Field-wise Word Frequency	Filter FPs
F3	Field-wise Word plus Tolerance	Filter FPs
G1	Graph Generation	Generate Graph
C1	Graph Expand	Connect Graph
C2	Operation Pair	Connect Graph

A Appendix

A.1 Post-processing Rules

Our post-processing has three types of rules shown in Table 3, i.e. filtering rules starting with F, graph reconstruction rules starting with G and graph connecting rules starting with C. These rules are plugins that can easily be extended with more.

Specifically, F1, F2, and F3 are filters based on global word frequency, field-wise word frequency, and field-wise word frequency plus a tolerance bound as explained in §3.4. G1 is a rule applied to generate an initial attack graph from detected log entries, which is similar to algorithms used to generate causal graphs from logs. C1 then connects disjoint graph components by expanding the graph and C2 pairs request and response operations in the graph.

A.2 ATLAS Investigation Cases.

We show four attack investigation cases (same cases in §3.5) generated by ATLAS in Figure 20. The yellow ones show the start point given by analyzers, the black ones indicate false positives, the shadowed black ones indicate the nodes that are misclassified as malicious, and the red ones represent the real attack nodes marked as malicious by ATLAS. For ATLAS, all edges and nodes that are directly connected with malicious nodes will be marked as malicious.

ATLAS can generate a precise attack story while missing key steps. As observed in the first case, ATLAS only predicts three attack entities when given a good attack clue as the start point, such as malicious IP Xaa.com. The small number of detected malicious entities allows ATLAS to generate precise attack history without introducing many false positives. However, it can miss key connections such as the edge between Firefox_12 and malicious IP 192.b.c.d.

In another kind of cases, ATLAS detects more attack entities while introducing many false positives. For example, ATLAS misclassifies a benign IP 172.d.e.f and process Search_1.exe in cases S2 and S4 as malicious, which will introduce all related benign behaviors of misclassified malicious entities as false positives. Overall, the performance of ATLAS depends on the given start points. However, choosing a good starting point can be challenging in real scenarios.

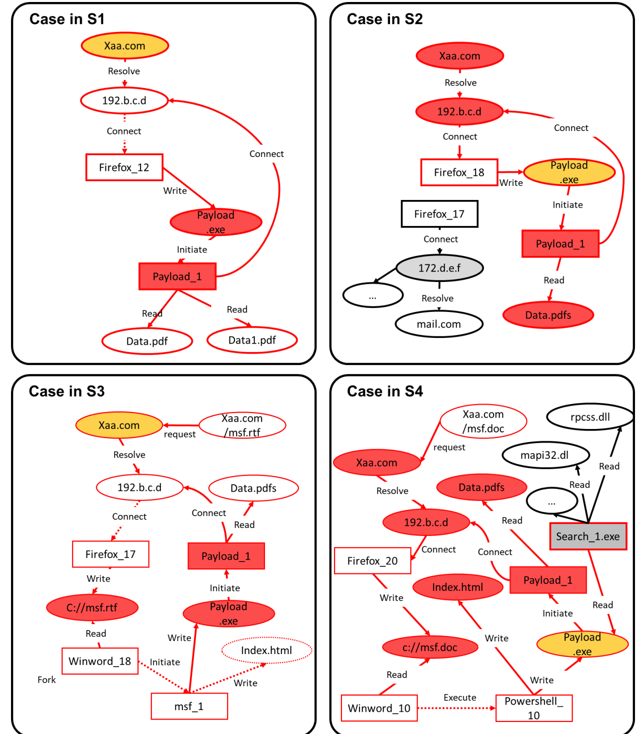


Figure 20: Investigation Cases by ATLAS.

A.3 Non-overlapping Attack Nodes

As mentioned in §3.1, all behaviors connected to an attack clue node, such as payload, are considered unsafe and malicious in ATLAS labeling. Such attack nodes are typical nodes that do not overlap with any benign behavior. We analyze the sensitivity to such non-overlapping attack nodes in terms of the number of these nodes and the changes in themselves.

Firstly, the number of such non-overlapping nodes is tightly related to the number of attack steps (e.g., using one less payload step reduces the number of associated payloads). Therefore, we use the evaluation of unsuccessful attack steps for our analysis (the number of such attack clues is shown in Table 1 and evaluation results are shown in Figure 10), where the first attack steps contain fewer non-overlapping attack nodes than the later steps. As shown in Figure 10, as the number of attack steps increases, we observe no significant change in TPR, but a decrease in FPR. This suggests that as the number of non-overlapping attack nodes increases, it is more beneficial for the model to distinguish the attack behavior. This is reasonable since more attack activities usually lead to more outliers and make the attacks easier to detect.

Secondly, we change the name of payload nodes to measure the sensitivity to changes in the attributes of non-overlapping nodes. As shown in Figure 17, the TPR and FPR are still very good and comparable to the original ones. Therefore, we do not think that changes in the non-overlapping attack nodes themselves will significantly affect AIRTAG.