Threat Behavior Textual Search by Attention Graph Isomorphism

Chanwoo Bae, Guanhong Tao, Zhuo Zhang, Xiangyu Zhang
Purdue University, USA
{bae68, taog, zhan3299}@purdue.edu, xyzhang@cs.purdue.edu

Abstract

Cyber attacks cause over $1 trillion loss every year. An important task for cyber security analysts is attack forensics. It entails understanding malware behaviors and attack origins. However, existing automated or manual malware analysis can only disclose a subset of behaviors due to inherent difficulties (e.g., malware cloaking and obfuscation). As such, analysts often resort to text search techniques to identify existing malware reports based on the symptoms they observe, exploiting the fact that malware samples share a lot of similarity, especially those from the same origin. In this paper, we propose a novel malware behavior search technique that is based on graph isomorphism at the attention layers of Transformer models. We also compose a large dataset collected from various agencies to facilitate such research. Our technique outperforms state-of-the-art methods, such as those based on sentence embeddings and keywords by 6-14%. In the case study of 10 real-world malwares, our technique can correctly attribute 8 of them to their ground truth origins while using Google only works for 3 cases.

1 Introduction

Cyber-attacks are a prominent threat to our daily life, causing over $1 trillion loss every year. Defending and mitigating cyber-attacks are hence critical. An important task in the arms race is attack forensics, which aims to determine malware behaviors, damages, and origins. It usually starts with an attack instance, e.g., a malware sample captured in the wild. The analysts use tools such as IDA (Ferguson and Kaminsky, 2008) to inspect its code body, and sand-boxing techniques such as Cuckoo (Oktavianto and Muhardianto, 2013) to execute it and observe its runtime behaviors. Attack forensics are important because the results can be used to assess damages and prevent future attacks. However, malware often employs sophisticated self-protection such as obfuscation (You and Yim, 2010) that changes code body to make it difficult to understand and/or masquerade a benign application, and cloaking that conceals malicious payload until certain (attack) conditions are satisfied. As a result, analysts usually can only disclose a part of malware behaviors. They hence heavily rely on text search to find existing related malware reports. Such search is usually driven by the observed behaviors such as sabotage, data exfiltration (regarding how they are performed?). The rationale is that cyber-attacks become increasingly organized (e.g., sponsored at a state-level), showing a substantial level of commonality in terms of the exploits used (i.e., bugs in target systems that allow the malware to penetrate), the payloads delivered, and their objectives, especially for those launched by a same *threat actor* (Malpedia) (i.e., an adversary or an organization of adversaries). As such, one can predict a new malware's full behaviors from reports of existing malware samples that share some commonality with the new sample. In fact, major security vendors have published a large volume of malware analysis reports. While they have the great potential to provide collective intelligence for future analysis, there has been an intrinsic barrier to fully leveraging such knowledge, namely, these threat reports are written in unstructured and informal natural languages. Since anyone can contribute such reports, it is difficult to standardize them.

Therefore, the key problem is a specialized text search challenge, which is called *cyber threat intelligence* (CTI) search following the terminology used in the domain. CTI search poses two main challenges; (i) supervised-learning is hardly feasible due to the lack of labeled datasets, and (ii) existing pre-trained general-purpose language models cannot effectively capture domain specific semantics. Sometimes, small changes for a general-purpose language model denote substantial semantic differences in CTI. For example, a *"file"* may describe either information stealing (*"**file** to leak*

Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics
Volume 1: Long Papers, pages 2616–2630

the stolen data from the program") or program exploitation (*"**file** to exploit the program for stealing data"*).

The most popular search method directly uses *indicators of compromise* (IoCs) of the malware sample, e.g., malware file hash (Catakoglu et al., 2016; Liao et al., 2016). It is the method used in VirusTotal (VirusTotal), a widely used malware analysis platform. Although using IoCs is precise and free from false positives, it cannot deal with the well known malware mutation problem (Liao et al., 2016) in which malware frequently and consistently changes its configurations, payloads, and even attack steps, to evade detection or simply update its functionalities. Another method is text similarity based malware behavior search. Existing text similarity methods largely fall into two categories, *keywords based* methods (Corley and Mihalcea, 2005; Harispe et al., 2015) and *sentence embedding based* methods (Le and Mikolov, 2014; Lau and Baldwin, 2016; Devlin et al., 2018; Reimers and Gurevych, 2019). The former focuses on domain specific keywords. It cannot effectively extract relations across keywords, which are critical in CTI search. In the above example, the keyword *"file"* needs to be analyzed with the relation of other words (i.e., *"leak"* or *"exploit"*) - keywords-based search (i.e., *"steal"*, *"program"*, *"data"*) will cause misunderstanding. In contrast, directly using embeddings tends to be unnecessarily distracted by the words that are not critical to CTI search.

We propose a novel CTI search technique. We collect a large repository of CTI reports from multiple agencies such as Kaspersky, Symantec and MacAfee. Specifically, Mitre ATT&CK (Mitre ATTACK) is a widely-known knowledge base of adversary techniques (i.e., behaviors) based on real-world malware observations. The repo covers reports in the past 20 years. We then use a *masked-language model* (MLM) based on Transformer to perform unsupervised learning on the dataset. We observe that the language model can pay special attention to IoC related words, and more importantly, their correlations. After training, instead of using the pre-trained embeddings, which are noisy due to the large natural language vocabulary, we construct a *attention graph* in which a node is a word token and an edge is introduced between two nodes when their attention is larger than a threshold. We then use graph similarity to determine CTI report similarity. We make the following contributions.

- We collect a large volume of existing CTI reports from reputable sources which could facilitate future research.
- We propose a novel attention graph based search method for CTI reports. It is particularly suitable in capturing the domain specific semantics of these reports.
- We compare our method with *doc2vec* (Le and Mikolov, 2014; Lau and Baldwin, 2016) (a sentence embedding based technique), keyword based text similarity methods (Corley and Mihalcea, 2005; Harispe et al., 2015), and a few state-of-the-art unsupervised learning based methods (Reimers and Gurevych, 2019). Our method consistently outperforms these baselines.
- In a case study of 10 real-world malware attacks, our search successfully finds the most relevant reports (from the past) that allow us to attribute 8 of these attacks to their true origins. In contrast, using *Google* can only correctly attributes 3 and a simple IoC-based search correctly attributes 2. One of the generative LLMs, GPT-4 (Google Bing) correctly answers 3.

2 Motivation

A real-world event in 2019 on a nuclear power plant in India (INDIA TODAY, 2019) illustrates how an information retrieval (i.e., our system) contributes to cyberattack investigations. It is a multi-stage attack that first penetrates some computers on the power-plant's network, leveraging a zero-day code vulnerability (e.g., a bug in browser) and then laid low and silently compromise more systems leveraging normal functionalities. The process may take days or even weeks. The payload was finally delivered, 3-5 days after the initial penetration, accessing the nuclear plant's confidential data. Such complex and multi-staged attacks are also called *advanced persistent threat* (APT) (Milajerdi et al., 2019). Assume a few days after the attack was initiated, security analysts noticed some system anomaly. Further assume they acquired an attack artifact, which is the executable malware file used in the first penetration step. From the file, analysts acquired the hash of the malware bfb39f486372a509...0364 and a few malware behaviors using a sandbox tool, namely, **(B-1)** *"use a dropper that has encrypted payload"*, **(B-2)** *"list all*

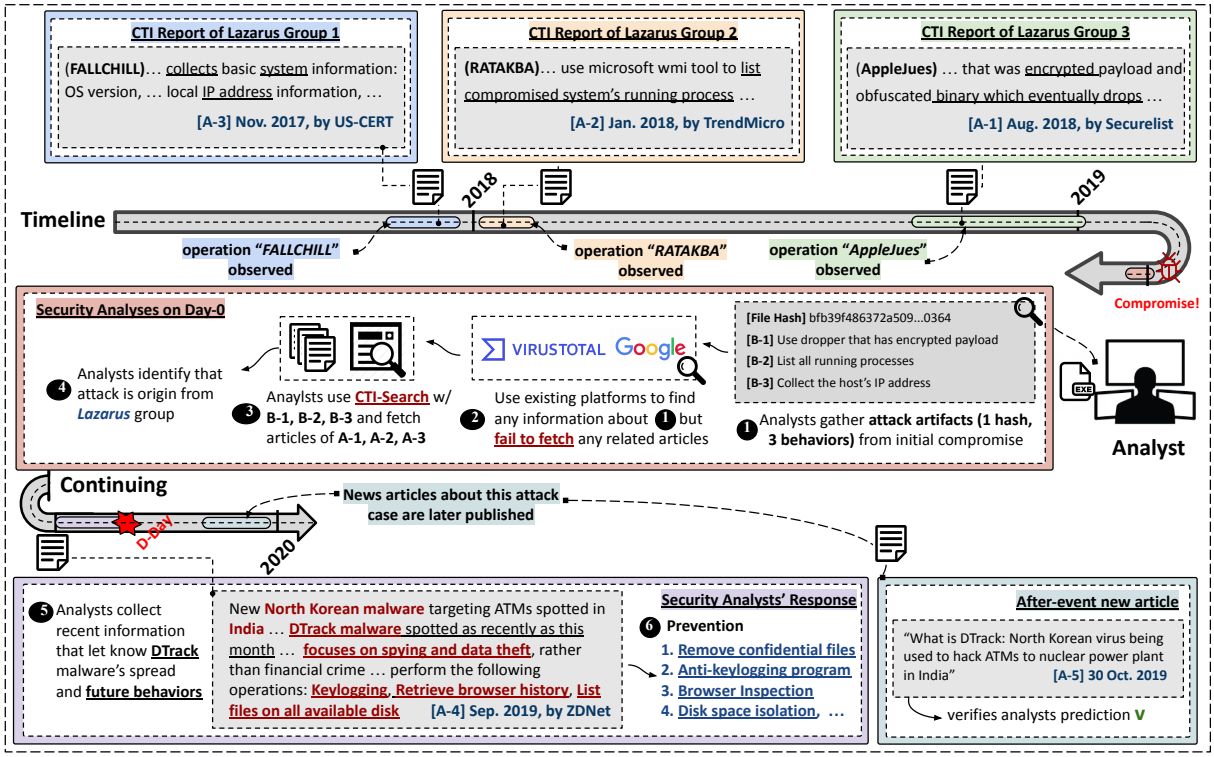

Figure 1: Motivation example: searching a real-world attack on an Indian nuclear plant. The arrow from left to right denotes the timeline. The attack happened in 2019 (the right-most spot on the timeline with a bug symbol). A few other attacks by the same threat actor were conducted before the 2019 attack and denoted by the blue, orange and green durations along the timeline. The large box "`Security Analyses on Day-0`" in the middle denotes the multiple methods the analyst could have used to analyze and search the attack. The boxes in the bottom show the real analysis reports of the attack there were produced long after the attack in 2020. Most of the information in those reports is covered by the *past* reports A-1, A-2, and A-3 retrieved by our method, illustrating that with our method, the attack could have been easily analyzed and attributed.

running processes", and **(B-3)** *"collect the host's IP address"*. This corresponds to step ❶ in Figure 1).

However, from these symptoms, the analysts can hardly determine the objective and scope of the attack. Since the power plant is critical infrastructure, they need answers to a number of questions, for example, what is the attack origin (is it from a major known threat actor)? and what are the attacker's ultimate interests (e.g., infrastructure sabotage and confidential information leak)? Critical decisions need to be made based on the answers to these questions.

The collected evidence is insufficient to answer these questions, which is very typical due to the inherent difficulties in malware analysis. The analysts usually resort to CTI search In our case, assume they first looked up the file hash on *VirusTotal* to check if the same attack has been conducted in the past. However, the attack was unique in 2019 and hence VirusTotal returned no match. In fact, the malware was first submitted to VirusTotal on October 28, 2019, one month after the initial attack. Then in step ❷, the analysts tried to find previous CTI reports using the behaviors, that is, B-1, B-2, and B-3. As the behaviors were written in natural language, they needed to rely on text search methods. Assume they used Google and the textual descriptions of the behaviors. However, the search results were not informative. Observe most of the retrieved items are not even semantically related. The semantically related items are in fact not related to the attack at all (refer to Appendix A.4).

Our Method. Assume that analysts had our search technique in 2019. Searching the three behaviors using our method, the analysts managed to retrieve 3 malware reports (i.e., A-1, A-2 and A-3) dated before the attack time which are all conducted by *Lazarus* group within last 2 years (see top three boxes in Figure 1). The analysts hence suspected the origin of attack was the *Lazarus*

group (step ❹). More importantly, recent threat intelligence article (**A-4** (ZDNet, 2019)) says that *Lazarus* groups recently perform the following attacks: (i) browser history collection, (ii) keylogging and (iii) disk-drive scrapping (step ❺). Therefore, the system administrator could employ the corresponding countermeasures (step ❻).

One month after the attack, real forensics reports were produced for the attack, named *Dtrack* (Economictimes, 2019). They indicated that the attack mainly focused on *stealing data from the keystroke (i.e., keyboard)*, *monitoring web-browsing history*, *data dump from local disk*. The information could have been disclosed by our technique much earlier if it was available at that time.

3 Related Work

Cyber Threat Intelligence Search. The most popular CTI search method uses IoC information (Liao et al., 2016). Existing work usually formulates the challenge as a *named entity recognition* (NER) problem, aiming to identify malware artifacts (e.g., *IP address* and *file hash*) from natural language documents (Liao et al., 2016; Zhu and Dumitras, 2018). Attack ontology was proposed in (Husari et al., 2017), aiming to formalize malware behaviors. Researchers have proposed to extend the datasources of threat intelligence, such as *Twitter* or *Darkweb* (Khandpur et al., 2017; Choshen et al., 2019; Wang et al., 2020; Jin et al., 2022).

Text Similarity. Similarly analysis is the key technique behind text search, which has been well studied (Corley and Mihalcea, 2005; Islam and Inkpen, 2008; Budanitsky and Hirst, 2006; Mihalcea et al., 2006; Ramage et al., 2009; Croce et al., 2011; Rahutomo et al., 2012; Kenter and De Rijke, 2015; Harispe et al., 2015; Rao et al., 2019). Basically, these approaches try to capture the common keywords between two texts. Some work adopts several optimization techniques; using word-weighting (Corley and Mihalcea, 2005; Kenter and De Rijke, 2015; Lopez-Gazpio et al., 2019) with IDF-score (Ramos et al., 2003), leveraging external knowledge (Islam and Inkpen, 2008; Budanitsky and Hirst, 2006) and use of word similarity methods (Corley and Mihalcea, 2005; Islam and Inkpen, 2008; Kenter and De Rijke, 2015; Harispe et al., 2015). In the CTI search, a knowledge-based approach is limited due to absence of such resources (e.g., *Wordnet* (Miller, 1995)). Meanwhile, word similarity and weighting are limited to the corpus-based approaches which are included as our baselines (Corley and Mihalcea, 2005; Harispe et al., 2015).

Recently, with the advances in AI, deep learning based approaches become increasingly popular (Tai et al., 2015; He and Lin, 2016; Wang et al., 2016; Devlin et al., 2018; Tien et al., 2019; Reimers and Gurevych, 2019; Sun et al., 2020). Specifically, embeddings are broadly in use with a number of popular training schemes: continuous bag of words (CBOW) based training (e.g., doc2vec (Le and Mikolov, 2014; Lau and Baldwin, 2016)) and masked language models (MLM) based training (Reimers and Gurevych, 2019; Devlin et al., 2018)

4 Design

CTI reports have domain specific semantics. For example, '*IP*' and '*network*' have similar meanings, '*drop*' and '*payload*' have strong correlations. Such semantics can hardly be captured by general-purpose language models. This challenge can be overcome using domain specific corpus during training. Therefore, a straightforward proposal is to use masked language model to train on a large corpus of CTI reports and then use sentence embeddings in CTI search. Specifically, a malware IoC is described by some sentence(s). The search can simply look for CTI reports that contain similar sentence embeddings. However, such a proposal can hardly work because distinct behaviors by small nuance differences between **B-1**, i.e., "*use a dropper that has encrypted payload*" and an entirely different behavior such as "*drops an encrypted payload*" will not be captured by embedding techniques. As of its ramifications, our later evaluation shows lower precision (i.e., false positives) by embedding techniques (Table 1). Furthermore, a CTI report may also describes a behavior using different sentence structures such that **A-1** reads "*The malware is an* **encrypted** *and* **obfuscated binary**, *..., it* **drops** *a piece of shell code ...*" which is more verbosely written. Although humans can easily determine that the corresponding malware has behavior **B-1**, the syntactic-based analysis (e.g., syntactic dependency analysis) can also fail.

We observe that the attention mechanism in Transformer models can capture (domain specific) semantic correlations between words. For example, there are strong correlations between '*dropper*' and "*encrypted payload*" in **B-1** and two strong corre-

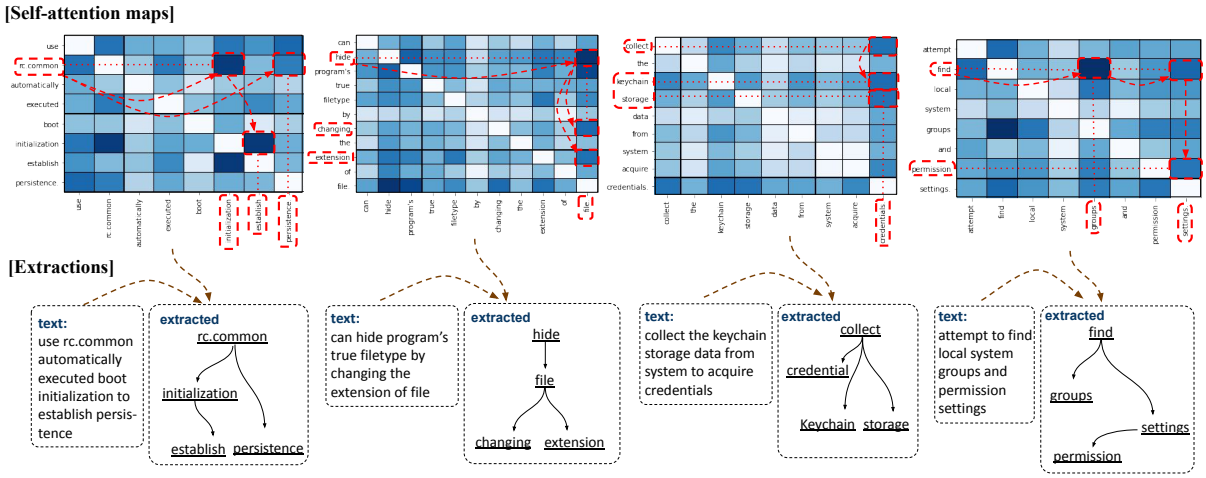

Figure 2: Figures show self-attention maps on examples (top). Based on word-to-word correlations in attention map, above examples show that we can extract the core representation of behaviors (bottom) from plain text.

lations in **B-3**: '*collect*' - '*host*' and '*host*' - '*IP address*'. These correlations are often not syntactic like verb-object relations. We depict the detail on how attention mechanism works with those examples (**A1**-**A3** and **B1**-**B3**) in Appendix A.4.

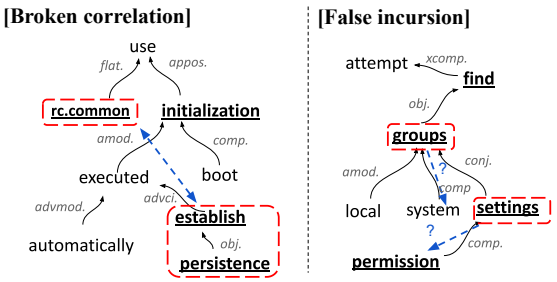

Figure 3: Inaccuracies in use of dependency trees (from two sentences in Figure 2). A dependency tree shows semantically correlated, but broken clauses (red boxes) due to no syntactic relation (left). Also it may incur false positive correlations (blue arrows) due to multiple equivalent neighbors (right).

It is believed that self-attentions map the word-to-word correlations in domain specific semantics. In this regard, our idea is to extract semantically structured graphs from text using self-attention maps. To the best of our knowledge, it is novel methodology to use attention mechanism for a semantic dependency parsing. Here, the graph construction is to traverse the sentence (i.e., set of words), prioritized by higher attention scores. Figure 2 illustrates how graphs can be constructed along with attention scores. Here, the key of semantic graph extraction is to abstract the core behaviors as a sub-graph form. All of above sentences are achieved abstractions by the self-attention guided exploration.

To compare attention maps with legacy (syntactic) dependency trees, we revisit two of above sentences with their parse trees (in Figure 3). It shows that the parser fails to capture the long-distance correlation in a lengthy text (i.e., broken correlation). The ramification is that it may need to include unnecessary words between two, i.e., incurring false positives. Also, syntactic relations may cause to explore the obsolete paths (i.e., false incursion).

Training and Use of Attentions. The benefit of our method is that the model uses self-supervised training. As myriad of security problems, the lack of labeled dataset crucially harm the model performances. Albeit it is feasible to collect a large scale of CTI text corpus, we have no supervision for training (i.e., no pairwise ground truth). In this regard, our method fits our domain problem in that it exploits the self-supervised learning. We hence use masked language model with a *BERT* (Devlin et al., 2018) architecture on the large-scale (8M words) CTI corpus (refer to Table 2).

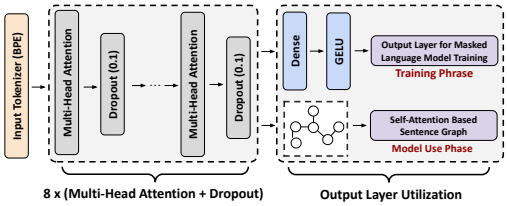

Figure 4: The utilization of self-attention for search.

The use of such model (i.e., pre-trained) is different from legacy that in LLMs (e.g., fine-tune) in that we construct the graphs by exploring the attention maps. We leverage the self-attention scores

to prioritize edges with higher attentions. In such method, it does not entail any supervised learning.

Sub-graph Matching and Similarity Score. After graph constructions, we use the sub-graph matching algorithm and the similarity score computation. We consider two words w_1 and w_2 match if and only if $|e(w_1) - e(w_2)| < \tau$ where $e(w)$ denotes the embedding of a word w and τ is a threshold. With the definition of matching nodes, our challenge can hence be reduced to the *subgraph isomorphism problem* (Sys et al., 1982), which aims to find the largest isomorphic sub-graph of two undirected graphs (Algorithm 1). The complexity of the problem is NP. However, since the graph for behavior description is small (10-15 nodes on average), the runtime is reasonable in practice, with our optimization of filtering irrelevant articles mentioned in Appendix A.2.

The similarity score of two isomorphic sub-graphs G_1 and G_2 is hence computed as follows.

$$\text{sim}(G_1, G_2) = \prod_{w_1, w_2 \in G_1 \times G_2} \left[\kappa(1 - |e(w_1) - e(w_2)|)\right]$$

where κ is a constant larger than 1, $e(w_1)$, and $e(w_2)$ the embeddings normalized to [0,1]. Note that κ needs to be larger than 1 such that large isomorphic sub-graphs yield a larger similarity score.

Implementation. We use 8 layers of multi-head attentions (of size 512) with a dropout on each layer (0.1 probability). Preprocessing is largely standard with some domain specific normalization for IoC related artifacts, such as IP. Specifically, we feed each CTI report under search to the model and acquire the self-attentions at the last layer. For each input text, we construct an *attention graph*, in which each node denotes a token. An edge is introduced between two nodes when their attention exceeds a threshold of 0.15. In sub-graph matching, we use 2.72, and 0.37 as κ and τ threshold each.

5 Evaluation

To train our models, we have collected 10,544 threat analysis articles from eight major security vendors (refer to Table 2). The corpus contains 500K sentences and 8M words. For the input tokenizing for self-attention layers, we use *byte-pair encoding* (Sennrich et al., 2015) and limit the number of tokens to 30,000 (originally, the dataset has 185K distinct words after lemmatization). We use *BERT* for the self-attention layers (with 20% of random masking and 2 epochs) and the *Gensim* framework to train a *word2vec* model with output vector size of 100 and 100 epochs for domain specific word embeddings.

Table 2: Pretraining Dataset

Vendor	# of Articles	# of Sents	# of Words
FireEye	843	50K	858K
Fortinet	541	49K	645K
IBM	926	43K	843K
Kaspersky	1,441	50K	858K
McAfee	626	38K	587K
Palo Alto	641	58K	897K
Symantec	177	16K	278K
ESET	5,349	149K	3M
Total	**10.5K**	**0.5M**	**8M**

Our evaluation answers follwing research questions: (**R1**) what is the effectiveness of the proposed method compared to existing techniques; (**R2**) how does the technique help real-world attack investigation; and (**R3**) how efficient is the method.

5.1 Effectiveness

We devise a controlled experiment to compare the precision and recall of different methods. We use the **SP-EVAL-SET-1** (Mitre ATTACK) and **SP-EVAL-SET-2** (CAPEC) datasets. Specifically, they provide attack behavior dictionaries such that for each threat behavior, they provide (i) a written description for the behavior and (ii) the associated real-world malware cases' descriptions. For each behavior, we construct a dataset as follows. We include all the malware cases associated with the behavior (the true positives) and the same number of random cases from other behaviors. In total, we test 423 behaviors from **SP-EVAL-SET-1**, 262 behaviors from **SP-EVAL-SET-2** and the aggregated number of cases are 14,096 and 2,002 for **SP-EVAL-SET-1** and **SP-EVAL-SET-2**, respectively.

We use the following baselines that are unsupervised learning based.

- ***Word Matching***: returning sentences based on the common words.
- ***Doc2Vec***: A sentence embedding technique based on the CBOW model (Le and Mikolov, 2014). It returns sentences based on embedding similarities.
- ***Transformer***: Training a *Transformer* model (Reimers and Gurevych, 2019) from scratch

Table 1: Effectiveness Evaluation (P stands for precision and R for recall)

Type	SP-EVAL-SET-1			SP-EVAL-SET-1		
	P.	R.	F1	P.	R.	F1
Simple Word Matching	0.59	0.97	0.73	0.60	0.98	0.75
Doc2Vec (Le and Mikolov, 2014)	0.69	0.77	0.73	0.66	0.81	0.73
Transformer (Reimers and Gurevych, 2019)	0.61	0.91	0.73	0.84	0.68	0.75
Transformer-Finetune (Turc et al., 2019)	0.62	0.89	0.73	0.65	0.91	0.76
Keyword Similarity 1 (Mihalcea et al., 2006)	0.73	0.90	0.80	0.73	0.88	0.79
Keyword Similarity 2 (Harispe et al., 2015)	0.60	0.95	0.74	0.55	0.98	0.71
Graph Isomorphism w/ Dependancy Parser	0.75	0.89	0.81	0.77	0.88	0.82
***Attention Graph Isomorphism** (Our)*	0.82	0.93	**0.87**	0.81	0.89	**0.85**

and using sentence embedding similarity.

- ***Transformer-Finetune***: Fine-tuning a pre-trained *BERT* model (BERT) and using embedding similarity.
- ***Keyword Similarity 1***: A widely used text similarity based method (Mihalcea et al., 2006) using word weights.
- ***Keyword Similarity 2***: A recent work prioritizing short texts, e.g., compact keywords (Kenter and De Rijke, 2015).
- ***Graph Isomorphism***: Our methodology.

All these results require a threshold to determine the retrieved cases. We try many thresholds and report the best results. The results are presented in Table 1. Observe our method (the last row) achieves the best performance. It has highest F1 score than any other baselines. Also observe that directly using sentence embeddings does not yield good results, neither do the keyword similarity based methods.

Analysis of Failing Cases. Table 3 shows a few failing cases by the baselines. In the first case, the embedding based methods yield the wrong results as the sentence embeddings are dominated by the verb such that the sentences with '*display*', '*find*' and '*identify*' are matched with the query sentence through the verb '*get*'. In comparison, the matching attention graphs by our method better disclose the essence. In the second case, the embedding based methods focus too much on the verbs. The keyword based methods report the wrong results because they find three keyword matches. However, these keywords do not have the semantic correlations as those in the true positive. We can observe the similar cases in the third.

5.2 Use in Real-world Attack Forensics

A critical task in forensics is to identify attack origins, that is, attributing attacks to their threat actors. In this experiment, we randomly select a few recent attacks and assume a subset of behaviors are known beforehand. We then use them to search the corresponding CTI reports. We use Google and IoC matching (similar to VirusTotal) as the baselines.

First, we collect additional CTI reports with explicit attack origin information and exclude all that overlap with the training set. The collection contains 258 articles with 12 major actors. The details are in Table 8 (Appendix). We then randomly gather 10 real-world attacks in *Mitre ATT&CK*. We use their behavior descriptions to search the article pool. We consider the actor with the largest number of matching as the attack actor.

Figure 5 represents the results. Our method successfully identifies 8 correct origins out of 10 cases. whereas Google identifies 3 correct answers.

To compare with IoC matching, we manually extract all IoC artifacts from the threat reports. It is common practice that authors attach such information), including the following types: *URL*, *IP*, *Hash*, *CVE*, *Registry*, *File* (IOC Parser). We exclude trivial *Windows* executable file names (e.g., cmd.exe) which cause a high volume of false positives. We then use exact matches of IoCs in the experiment. As a result, only two attacks (*Winnti* and *RDAT*) can be attributed to their origins.

We also test the SOTA internet-connected LLM, namely GPT-4 (i.e., Bing Chat) by prompting to extract related articles. GPT-4 only answers for 5 origins. Among those 5, it can correctly attribute 3 (*Dtrack*, *HotCroissant* and *KerrDown*).

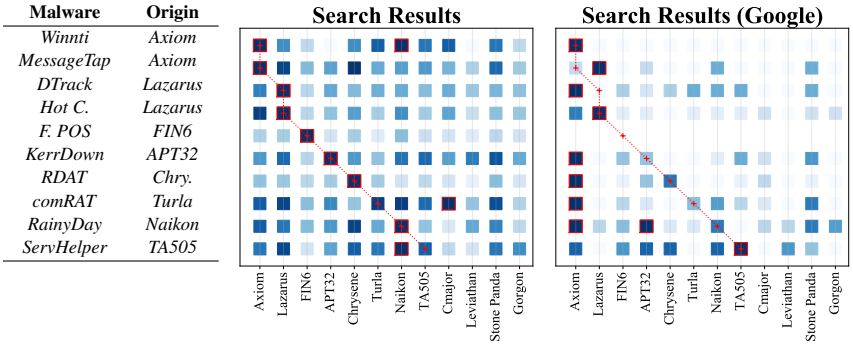

Figure 5: Search Result for Attack Origin Identification. In each figure, a red cross on the line (+—+—+) denotes the target origin of a malware. In each row (i.e., a malware), the actor with the largest number of search results is marked by a red square (□). Therefore, a co-location of the two symbols tells the success of origin identification (⊞).

5.3 Efficiency

We have implemented non-lossy search optimizations; (i) graph caching and (ii) sentences clustering and evaluated the runtime efficiency of our method. Our optimizations, evaluation and system specification details can be found in Appendix A.2.

Table 4: Result on Efficiency Test

Type	Search Space		
	20K	50K	100K
Baseline (matching)	20s	53s	1m45s
Our System (optimized)	09s	28s	57s

We compare our system (after fully optimized) to the simplest word matching in efficiency. The test performs the search query of 10 words by varying search space sizes. The baseline includes a text preprocessing and word-to-word comparison within a pair of two sentences.

Before any optimizations, a raw implementation of the graph isomorphism costs ∼17m with 20K search space, ∼4h with 100K (Table 6). However, our non-lossy optimization drastically reduces search times as shown in Table 4 that become comparable to the baseline.

6 Discussion

Word Embedding. One can benefit from contextualized word embedding. Also, the *Transformer* model contains pre-trained token embedding that can be used to measure word to word distance. It is worth considering a use of such embedding methodologies. In our study, we use *word2vec* in that attack describing terms (e.g., exploit, encrypt) tend to be absolute in different contexts.

Traversal with Self-Attentions. While we use a threshold to construct graphs from text (Algorithm 1), the methodology is not limited in general. For example, one can use a traversal algorithm prioritized by attention scores. Attention map may also help to build syntax parsers as it retrieve semantic correlations in sentences.

Dataset Release. We publish our dataest[1]. Any following work must be only on research purposes. It is worth noting that our dataset is a collection of open articles from various vendors. As our dataset is up to year of 2023, one may need to reproduce the collection by following the instruction.

7 Conclusion

We propose a novel CTI search method using attention graph isomorphism. We have shown that our method improves the effectiveness of CTI search for comparative evaluations. Our case study also shows that it drastically improves attack origin identification. Our technique can correctly attribute 8 of 10 recent attacks while Google only attributes 3.

8 Limitations

Since our system resorts to word-level embeddings, it has difficulty handling cases in which a word is equivalent to a phrase. For example, *"obfuscate"* and *"make it difficult to understand"* are semantically similar. But such similarity may not be captured by our technique. We speculate with a large corpus, the embeddings by transformer can better consider the context and hence capture the similarity.

[1] https://github.com/cwbae10-purdue/cti-eacl24.git

Table 3: Failing cases of baselines. Matching background colors denote matching words (based on embeddings). The underlined words in the query and the sentence returned by our method form the matching attention graphs. Our method returns the correct CTIs in all these cases.

[Query] ***get*** *information about* ***running*** ***processes*** *on system*
[D2V] *Sykipot may use netstat to display active network connections* **[Transformer]** *GravityRAT uses netstat to find open ports on victim's machine* **[Transformer]** *Get2 has ability to identify current username of infected host* **[Our Search Result]** *PowerShower has ability to ... module to* ***retrieve*** *list of* ***active*** ***processes***
[Query] ***execute*** *their own malicious* ***payloads*** *by* ***hijacking*** ***library*** *manifest used to load DLLs*
[D2V, Transformer] *APT28 has used tools to perform keylogging* **[D2V]** *BADNEWS is capable of executing commands via cmd.exe* **[Keyword Similarity]** *SeaDuke uses module to execute Mimikatz with PowerShell to perform Pass Ticket* **[Keyword Similarity]** *PowerSploit modules are written in and executed via PowerShell* **[Our Search Result]** *HyperBro has used legitimate application to* ***sideload*** ***DLL*** *to decrypt decompress and* ***run*** ***payload****.*
[Query] *with no prior knowledge of legitimate credentials within system or environment,* ***guess*** ***passwords*** *to attempt* ***access*** *to* ***accounts***
[D2V, Transformer] *Zeus Panda checks to see if anti virus anti spyware or firewall products are installed in victim's environment* **[Keyword Similarity]** *Agent Tesla can collect system's computer name and also has capability to collect information on processor memory and video card* **[Our Search Result]** *SpeakUp can perform* ***bruteforce*** *using predefined list of usernames and* ***passwords*** *in attempt to* ***log-in*** *to* ***administrative*** *panels*

9 Ethnics Statements

Our system resorts to cyber threat intelligence (CTI) dataset. This may impose the risk factors as follows;

- ***Possible Exposure of Threat Knowledge***: As dataset comprise threat analysis articles, it may contain potential risks to be abused.
- ***Adversarial Use of Knowledge***: Attackers may use the information retrieval system to operate advanced attacks or avoid possible defenses assisted by threat intelligence.
- ***Concerns on Privacy Information***: Threat knowledge contained in the dataset may hold real cyberattack cases. It may contain statewide or private damages or loss which can lead to violation of privacy.

We have only used the dataest for research purposes (i.e., textual analysis). All reproduced work from our dataset must be on same objectives. In this regard, good practices as below need to be observed;

- ***Avoiding Use of Recent Critical Knowledge***: One must refrain from the use of on-going (or recent) threat information as it might aggravate the circumstances.
- ***Avoiding Use of Effective Vulnerabilities***: If threat knowledge is still effective (e.g., before patch), one must not include such information.
- ***Excluding Privacy Information on Damages***: One must refrain from including privacy damages (e.g., loss/damage of specific institutions) to technical articles (e.g., case studies). It may contain privacy information.

References

BERT. https://github.com/google-research/bert. Accessed: 2022-06-20.

Alexander Budanitsky and Graeme Hirst. 2006. Evaluating wordnet-based measures of lexical semantic relatedness. *Computational linguistics*, 32(1):13–47.

CAPEC. https://capec.mitre.org/. Accessed: 2022-06-20.

Onur Catakoglu, Marco Balduzzi, and Davide Balzarotti. 2016. Automatic extraction of indicators of compromise for web applications. In *Proceedings of the 25th international conference on world wide web*, pages 333–343.

Danqi Chen and Christopher D Manning. 2014. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 740–750.

Leshem Choshen, Dan Eldad, Daniel Hershcovich, Elior Sulem, and Omri Abend. 2019. The language of legal and illegal activity on the darknet. *arXiv preprint arXiv:1905.05543*.

Courtney D Corley and Rada Mihalcea. 2005. Measuring the semantic similarity of texts. In *Proceedings of the ACL workshop on empirical modeling of semantic equivalence and entailment*, pages 13–18.

Danilo Croce, Alessandro Moschitti, and Roberto Basili. 2011. Structured lexical similarity via convolution kernels on dependency trees. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 1034–1046.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Economictimes. 2019. What is dtrack: North korean virus being used to hack atms to nuclear power plant in india. Published: 2019-10-22.

Justin Ferguson and Dan Kaminsky. 2008. *Reverse engineering code with IDA Pro*. Syngress.

Peng Gao, Fei Shao, Xiaoyuan Liu, Xusheng Xiao, Zheng Qin, Fengyuan Xu, Prateek Mittal, Sanjeev R Kulkarni, and Dawn Song. 2021. Enabling efficient cyber threat hunting with cyber threat intelligence. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pages 193–204. IEEE.

Sébastien Harispe, Sylvie Ranwez, Stefan Janaqi, and Jacky Montmain. 2015. Semantic similarity from natural language and ontology analysis. *Synthesis Lectures on Human Language Technologies*, 8(1):1–254.

Hua He and Jimmy Lin. 2016. Pairwise word interaction modeling with deep neural networks for semantic similarity measurement. In *Proceedings of the 2016 conference of the north American chapter of the Association for Computational Linguistics: human language technologies*, pages 937–948.

Ghaith Husari, Ehab Al-Shaer, Mohiuddin Ahmed, Bill Chu, and Xi Niu. 2017. Ttpdrill: Automatic and accurate extraction of threat actions from unstructured text of cti sources. In *Proceedings of the 33rd Annual Computer Security Applications Conference*, pages 103–115.

IDA. https://hex-rays.com/. Accessed: 2022-06-20.

INDIA TODAY. 2019. What is dtrack: North korean virus being used to hack atms to nuclear power plant in india. Published: 2019-10-30.

IOC Parser. https://github.com/PaloAltoNetworks/ioc-parser. Accessed: 2022-06-20.

Aminul Islam and Diana Inkpen. 2008. Semantic text similarity using corpus-based word similarity and string similarity. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 2(2):1–25.

Youngjin Jin, Eugene Jang, Yongjae Lee, Seungwon Shin, and Jin-Woo Chung. 2022. Shedding new light on the language of the dark web. *arXiv preprint arXiv:2204.06885*.

Tom Kenter and Maarten De Rijke. 2015. Short text similarity with word embeddings. In *Proceedings of the 24th ACM international on conference on information and knowledge management*, pages 1411–1420.

Rupinder Paul Khandpur, Taoran Ji, Steve Jan, Gang Wang, Chang-Tien Lu, and Naren Ramakrishnan. 2017. Crowdsourcing cybersecurity: Cyber attack detection using social media. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 1049–1057.

Jey Han Lau and Timothy Baldwin. 2016. An empirical evaluation of doc2vec with practical insights into document embedding generation. *arXiv preprint arXiv:1607.05368*.

Quoc Le and Tomas Mikolov. 2014. Distributed representations of sentences and documents. In *International conference on machine learning*, pages 1188–1196. PMLR.

Xiaojing Liao, Kan Yuan, XiaoFeng Wang, Zhou Li, Luyi Xing, and Raheem Beyah. 2016. Acing the ioc game: Toward automatic discovery and analysis of open-source cyber threat intelligence. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 755–766.

Inigo Lopez-Gazpio, Montse Maritxalar, Mirella Lapata, and Eneko Agirre. 2019. Word n-gram attention models for sentence similarity and inference. *Expert Systems with Applications*, 132:1–11.

Malpedia. https://malpedia.caad.fkie.fraunhofer.de/. Accessed: 2022-06-20.

Oren Melamud, Jacob Goldberger, and Ido Dagan. 2016. context2vec: Learning generic context embedding with bidirectional lstm. In *Proceedings of the 20th SIGNLL conference on computational natural language learning*, pages 51–61.

Rada Mihalcea, Courtney Corley, Carlo Strapparava, et al. 2006. Corpus-based and knowledge-based measures of text semantic similarity. In *Aaai*, volume 6, pages 775–780.

Sadegh M Milajerdi, Rigel Gjomemo, Birhanu Eshete, Ramachandran Sekar, and VN Venkatakrishnan. 2019. Holmes: real-time apt detection through correlation of suspicious information flows. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1137–1152. IEEE.

George A Miller. 1995. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41.

Mitre ATTACK. https://attack.mitre.org/. Accessed: 2022-06-20.

NLTK. https://www.nltk.org/. Accessed: 2022-06-20.

Digit Oktavianto and Iqbal Muhardianto. 2013. *Cuckoo malware analysis*. Packt Publishing Ltd.

Faisal Rahutomo, Teruaki Kitasuka, and Masayoshi Aritsugi. 2012. Semantic cosine similarity. In *The 7th international student conference on advanced science and technology ICAST*, volume 4, page 1.

Daniel Ramage, Anna N Rafferty, and Christopher D Manning. 2009. Random walks for text semantic similarity. In *Proceedings of the 2009 workshop on graph-based methods for natural language processing (TextGraphs-4)*, pages 23–31.

Juan Ramos et al. 2003. Using tf-idf to determine word relevance in document queries. In *Proceedings of the first instructional conference on machine learning*, volume 242, pages 29–48. New Jersey, USA.

Jinfeng Rao, Linqing Liu, Yi Tay, Wei Yang, Peng Shi, and Jimmy Lin. 2019. Bridging the gap between relevance matching and semantic matching for short text similarity modeling. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5370–5381.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*.

Securelist. 2018. Operation applejeus: Lazarus hits cryptocurrency exchange with fake installer and macos malware. Published: 2018-08-23.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2015. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*.

Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. 2019. Mass: Masked sequence to sequence pre-training for language generation. *arXiv preprint arXiv:1905.02450*.

Yu Sun, Shuohuan Wang, Yukun Li, Shikun Feng, Hao Tian, Hua Wu, and Haifeng Wang. 2020. Ernie 2.0: A continual pre-training framework for language understanding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 8968–8975.

Maciej M Sys et al. 1982. The subgraph isomorphism problem for outerplanar graphs. *Theoretical Computer Science*, 17(1):91–97.

Kai Sheng Tai, Richard Socher, and Christopher D Manning. 2015. Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075*.

TechNadu. 2019. The lazarus group is using a new banking malware against indian banks.

Nguyen Huy Tien, Nguyen Minh Le, Yamasaki Tomohiro, and Izuha Tatsuya. 2019. Sentence modeling via multiple word embeddings and multi-level comparison for semantic textual similarity. *Information Processing & Management*, 56(6):102090.

TrendMicro. 2018. Lazarus campaign uses remote tools, ratankba, and more. Published: 2018-01-24.

Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Well-read students learn better: On the importance of pre-training compact models. *arXiv preprint arXiv:1908.08962*.

US-CERT. 2017. Hidden cobra – north korean remote administration tool: Fallchill. Published: 2018-08-23.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

VirusTotal. `https://www.virustotal.com/`. Accessed: 2022-06-20.

Peng Wang Wang, Xiaojing Liao Liao, Yue Qin, and XiaoFeng Wang. 2020. Into the deep web: Understanding e-commercefraud from autonomous chat with cybercriminals. In *Proceedings of the ISOC Network and Distributed System Security Symposium (NDSS), 2020*.

Zhiguo Wang, Haitao Mi, and Abraham Ittycheriah. 2016. Sentence similarity learning by lexical decomposition and composition. *arXiv preprint arXiv:1602.07019*.

David Yenicelik, Florian Schmidt, and Yannic Kilcher. 2020. How does bert capture semantics? a closer look at polysemous words. In *Proceedings of the Third BlackboxNLP Workshop on Analyzing and Interpreting Neural Networks for NLP*, pages 156–162.

Ilsun You and Kangbin Yim. 2010. Malware obfuscation techniques: A brief survey. In *2010 International conference on broadband, wireless computing, communication and applications*, pages 297–300. IEEE.

ZDNet. 2019. New north korean malware targeting atms spotted in india.

Eugenia Lostri James A. Lewis Zhanna Malekos Smith. 2020. The hidden costs of cybercrime. Accessed: 2017-11-14.

Ziyun Zhu and Tudor Dumitras. 2018. Chainsmith: Automatically learning the semantics of malicious campaigns by mining threat intelligence reports. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 458–472. IEEE.

A Appendix

A.1 Ablation Study

We conduct an ablation study in that;

(i) we first eliminate a self-attention based graph builder and connect all possible pairs (make sentence a fully-connected-graph) and run isomorphic sub-graph discovery to get similarity score,

(ii) we do not use *Word2Vec* model so that isomorphic sub-graph cannot tolerate similar, but different words, therefore, it is only able to match same words.

Table 5: Result on Ablation Study

Type	P.	R.	F1	Δ F1
w/o Self-Attention	.75	.78	.76	-12.64%
w/o Word2Vec	.82	.82	.82	-5.75%

Under this configuration, we revisit the search performance evaluation on **SP-EVAL-SET-1**. For each modification, we set the threshold value again which maximizes the **F1**-score. Table 5 shows the result of ablation test. The "**Loss of F1**" column says the loss of **F1** score in percentile from our original result (note that our original method scores 87% of **F1**-score).

A.2 Search Query Time

Before we measure the query-time performance, we introduce two optimization techniques; (i) graph caching (GC) and (ii) sentences clustering (SC);

We notice that our self-attention based graph builder is a bottleneck against time performance, in average, costs 0.5s for a single sentence (while isomorphic sub-graph discovery module takes 5ms in average which is almost zero relatively). However, graph building for search space sentences does not need to be processed on-the-fly. Therefore, we use the graph caching (GC) optimization where we pre-build graphs from the sentences and cache to the database.

Also, we use a lossless optimization method, namely, sentence clustering (SC) in that we filter our any of non-related sentences have no synonym from the query sentence (recall that synonym is defined as two words those embedding vectors are within the constant τ-distance). For that, we pre-build a map from every word in dictionary (185K words) to sentences in search space (i.e., sentence clustering by word that includes all sentences which hold at least one of its synonyms). When a query sentence is given, we extract words from it and load their sentence-clusters - therefore, sentences belong to those clusters become the reduced search space.

Here, we measure searching time. We run our system with an Intel Xeon 2.20GHz CPU and 196GB memory space. We pick 20K, 50K and 100K of random sentences from our threat report corpus as for a search space, and randomly pick 5-word, 10-word size query sentences (10 sentences per each and measure their average/minimum/maximum). Then, we measure the results from original search (w/o **OPT**), search with **GC** and both-enabled, i.e., **GC+SC**.

Table 6: Query Time Measurement (**a.**:average, **m.**:min, **M.**:max)

Type		Search Space (# Sentences) 20K	50K	100K
Baseline (matching)	**a.**	20s	53s	1m45s
	m.	20s	52s	1m43s
	M.	21s	54s	1m46s
w/o **OPT**	**a.**	17m03s	43m49s	4h16m
	m.	16m45s	42m57s	4h14m
	M.	17m15s	17m15s	4h17m
w/ **GC**	**a.**	28s	24s	2m44s
	m.	11s	32s	1m04s
	M.	44s	41s	3m55s
w/ **GC+SC**	**a.**	05s	14s	34s
	m.	00s	02s	05s
	M.	14s	18s	1m19s

(a) **query size: 5 words**

Type		Search Space (# Sentences) 20K	50K	100K
Baseline (matching)	**a.**	20s	53s	1m45s
	m.	20s	53s	1m44s
	M.	21s	54s	1m47s
w/o **OPT**	**a.**	17m58s	46m30s	4h21m
	m.	17m27s	45m01s	4h18m
	M.	2m46s	18m46s	4h25m
w/ **GC**	**a.**	1m23s	4m05s	8m03s
	m.	53s	2m36s	5m03s
	M.	2m12s	2m12s	12m41s
w/ **GC+SC**	**a.**	09s	28s	57s
	m.	03s	11s	25s
	M.	23s	23s	2m21s

(b) **query size: 10 words**

Without any optimizations, a query takes from 16m up to 4h based on search space. However, this can be drastically reduced by our optimization scheme.

If we use the **GC** method, a query time becomes less 10 minutes, and, with **SC** method enabled, it is expected to be around a few minutes or less than a minute. Note that 100K of search space is not trivial. We also claim that our system is able to be distributed to multiple machines by dividing the search space.

A.3 Dataset Tables

This section holds additional tables for our dataset.

Table 7: Malware Bahaviors and IoCs Set (**OI-EVAL-SET-MALWARE**). **B.** stands for Behaviors

Malware	Actor	# of B.	# of IOCs
Winnti	*Axiom*	3	123
MessageTap	*Axiom*	7	3
DTrack	*Lazarus*	15	25
HotCroissant	*Lazarus*	15	14
FrameworkPOS	*FIN6*	5	25
KerrDown	*APT32*	7	49
rDAT	*Chrysene*	16	6
comRAT	*Turla*	16	16
RainyDay	*Naikon*	16	10
ServHelper	*TA505*	8	92
Total	-	62	363

Table 8: Attack Origin Identifying Evaluation Set (**OI-EVAL-SET-ACTOR**)

Actor	# of Articles	# of Sentences	# of Words
FIN6	17	1,280	21K
Leviathan	17	1,060	18K
Axiom	21	2,361	38K
Stone Panda	17	1,333	23K
Lazarus	29	2,338	37K
Gorgon	22	1,549	23K
Turla	24	1,646	28K
TA505	27	2,310	34K
Chrysene	21	1,400	25K
APT32	20	1,430	26K
Naikon	21	1,109	20K
C-Major	22	1,354	23K
Total	**258**	**16K**	**227K**

A.4 Continued from Motivation

How Our System Works. Figure 6 illustrates how our method works on the motivation example. For each behavior, the second, third, and fourth columns show the attention graph for the IoC description, the graph for the relevant sentence(s) in the corresponding CTI report, and the matched subgraphs. For example in **B-1**, our method matches the subgraph including '*dropper*', '*encrypted*', and '*payload'* in the IoC description to that in the report including '*drop*', '*encrypted*', and '*binary*'. Note that in the context of attack forensics, '*binary*' is a noun meaning a binary executable file which may be a payload on it own or include a payload. Therefore, our trained language model produces close embeddings for the two. There are similar sub-graph matches for **B-2** and **B-3** as well. □

Google Search. Assume the analyst searches **B-1**, **B-2** and **B-3** on Google. Most of top search results are not informative. It prioritizes instructional forums due to page ranking and does not bring **A-1**, **A-2** and **A-3**.

In fact, subtle query differences affect Google search results. Our investigation shows that Google fetches the articles (i.e., **A-1**, **A-2**, **A-3**) only if they query "*encrypted binary that **eventually** drops*", "*use **wmi too binary** list running processes*" and "*collect **infected** machine IP addresses*", respectively which require specific keywords overlapping (words in bold) to retrieve corresponding articles. It also requires tedious scrolling to find them. □

Algorithm 1 An algorithm to discover an isomorphic sub-graphs

// V_1 and V_2 have vector nodes (i.e., embedded).
$G_1 \leftarrow G(V_1, E_1)$, $G_2 \leftarrow G(V_2, E_2)$
$S \leftarrow \emptyset$, $\tau > 0$

for $v \in V_1$ and $w \in V_2$ s.t. $|v - w| < \tau$ **do**
  match $\leftarrow$ false
  while $s \in S$ **do**
    while $(w', v') \in s$ **do**
      if $(w \in \text{Neighbors}(w')) \wedge (v \in \text{Neighbors}(v'))$ **then**
        s.add((w, v))
        match $\leftarrow$ true
      end if
    end while
  end while
  if match $\neq$ true **then**
    S.add($\{(w, v)\}$)
  end if
end for

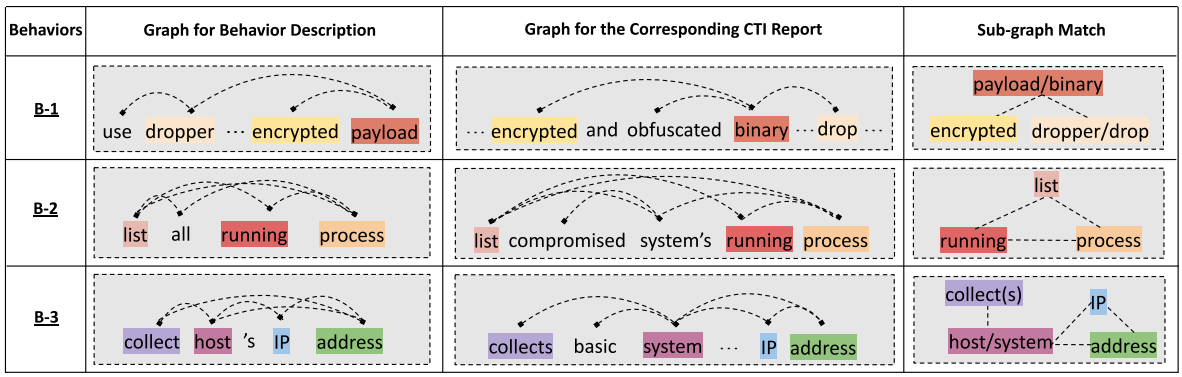

Figure 6: Attention graphs for the motivation example