GapFinder: Finding Inconsistency of Security Information from Unstructured Text

Hyeonseong Jo† Jinwoo Kim† Phillip Porras‡ Vinod Yegneswaran‡ Seungwon Shin†
† KAIST ‡SRI International
{hsjjo, jinwoo.kim, claude}@kaist.ac.kr {porras, vinod}@csl.sri.com

***Abstract*—Textual data mining of open source intelligence on the Web has become an increasingly important topic across a wide range of domains such as business, law enforcement, military, and cybersecurity. Text mining efforts utilize natural language processing to transform unstructured web content into structured forms that can drive various machine learning applications and data indexing services. For example, applications for text mining in cybersecurity have produced a range of threat intelligence services that serve the IT industry. However, a less studied problem is that of automating the identification of semantic inconsistencies among various text input sources. In this paper, we introduce GapFinder, a new inconsistency checking system for identifying semantic inconsistencies within the cybersecurity domain. Specifically, we examine the problem of identifying technical inconsistencies that arise in the functional descriptions of open source malware threat reporting information. Our evaluation, using tens of thousands of relations derived from web-based malware threat reports, demonstrates the ability of GapFinder to identify the presence of inconsistencies.**

***Index Terms*—Cyber Threat Intelligence, CTI, Inconsistency**

I. Introduction

As the frequency and sophistication of cyber attacks continue to rise, so too has grown the interest in services that produce cyber threat intelligence (CTI). Recently, several research projects have proposed CTI related frameworks and approaches [16], [21] to automatically analyze and identify advanced attacks. For example, TTPDrill suggested a way to analyze reported attack scenarios and derive possible defense methods automatically [16]. In addition, Liao et al. proposed an approach to analyzing indicators of compromise (IOC) automatically, and then used this information to derive efficient methods for understanding diverse attack scenarios [21].

One underlying commonality among these and other proposed CTI methods is a central reliance on publicly available information, such as blog articles, research papers, and security reports. Indeed, the timeliness and accuracy of these CTI methods (including their detection rate and accuracy) are a function of the quality of the web-mined information that they invest. As such, an informed reliance on these services poses an important question: *Can one derive a quality metric that captures the consistency and accuracy of the web-mined information used to drive a CTI service?* This question is the starting point of our research work.

If we build a CTI system based on open source web content, the most critical consideration is that of establishing, where possible, the correctness of this information. At present, however, automated consistency analysis for the correctness of open information is not seriously considered when a CTI system is built. This paper explores an approach that identifies correctness metrics within the domain of open source intelligence reports regarding malware functional descriptions from security blog crawling systems. For example, consider a crawler that collects information about the *Shylock* malware from two different security blog sites A[1] and B[2]. One text segment from site A reports that *Initially discovered in February 2011 by security firm Trusteer, Shylock delivers*, while another text segment from B reports that *The Shylock malware, named after a character from Shakespeare's The Merchant of Venice, was first discovered in September 2011*. While both text segments include a Shylock reference, each mentions a different discovery date (February 2011 vs., September 2011). An ideal CTI system would be able to detect the inconsistency, and if possible, determine which discovery date is most likely to be accurate.

Verifying the correctness of information is a challenging research topic, for which some schemes have been proposed [36], [38], [7], [37] in some text mining domains. However, existing schemes cannot be directly applied to vetting security specific information for the following three reasons. First, these existing schemes focus on verifying the correctness between already-structured (formatted) data (e.g., height of mountain, authors of a book), and they do not apply to evaluating *unstructured* data. Unfortunately, the open source information used to drive CTI services relies on unstructured web text, written in natural language [13]. Therefore, some specific language processing techniques, such as named entity recognition (NER) and relation extraction (RE), for the cybersecurity domain are needed to extract structured values for security information. Second, security information extracted from unstructured texts still requires additional formalization. If we extract data (e.g., nouns and verbs) from unstructured texts, many different shapes of terms represent the same meaning. For example, *email* and *attachment* are different terms, but in terms of the infection method of malware, both terms are based on the same technique, *phishing*. Thus, we need to refine extracted data to normalize their forms or make a dictionary for thesaurus of security terms. Third, while one can derive a thesaurus of some security terms, other relevant terms are more difficult to isolate semantic relations, such as

[1] www.scmagazine.com/home/security-news/shylock-banking-malware-can-detect-remote-desktops/

[2] www.ehackingnews.com/2012/08/shylock-trojan-injects-attackers-phone.html

for malware names. Each malware may have several aliases to refer to itself due to the different naming conventions of antivirus (AV) vendors. For example, Conficker worm has been assigned various parallel names from competing teams that published independent analyses: Conficker (by Sophos, ESET), Downadup (by Symantec), and Kido (by Kaspersky). Therefore, we need to understand whether some specific security terms are related to each other.

This paper proposes a systematic approach to detecting inconsistencies in security information from multiple publicly available sources. Our system, GapFinder, deconstructs this problem into three main tasks: (i) the extraction of structured data from unstructured text, (ii) a data refinement process on the extracted data, and (iii) the formulation of semantic connections between malware aliases. First, the graph constructor, consisting of an entity tagger, relation builder, and relation expander, derives security specific structured data from unstructured text with consideration for cybersecurity specific issues. For the extracted data, the graph constructor builds malware graphs, which consist only of relations that contain the same malware name. Second, the data formatter performs a data refinement process on the data within the malware graphs. The data extracted from unstructured text are expressed with diverse shapes of terms to convey similar meanings. To address this issue, the data refinement process normalizes different terms with similar or identical meanings into the same expression. Third, the malware graphs, based on a single malware name, are disconnected from each other. To compare the data in the malware graphs that have different names but indicate the same malware, the alias connector produces connections between malware graphs that have alias relationships, such as between Conficker and Kido.

Our evaluation, using tens of thousands of relations derived from malware threat reports, demonstrates the ability of GapFinder to identify the presence of inconsistencies. First, most of the sources do not cover all aspects of the malware description, requiring one to produce a robust description of the malware through an aggregation of descriptions captured among multiple sources. For example, Locky ransomware uses five infection methods to propagate itself, and at least three sources are needed to capture the full spectrum of its infection methods. Second, many sources have made various claims about the first detection date of malware, diverging from 1 month to 7 months. For example, while four sources claim that Shylock was detected at February 2011, seven sources insist that the malware was discovered in September 2011.

In summary, the contributions of our work are as follows:

- We present GapFinder that derives structured relations from unstructured text about the four main features of malware.
- GapFinder addresses the inconsistency between any forms of entities, structured or unstructured text, by inferring common terms for different words with similar or equivalent meaning.
- We evaluate GapFinder against 470K security reports and find many inconsistencies in the four main features of malware.

II. Motivation and Term Definition

A. Motivating Example

As cybersecurity incidents continue to grow in both volume and sophistication [17], defenders now regularly respond to these incidents with the publication of CTI reports. CTI reports capture what is known about an incident, commonly from a defensive perspective. Several previous studies have used these reports to extract IOCs (e.g., malware signatures, botnet IPs) [21], [3] or to develop automated and context-aware CTI analytics [16]. While CTI analytics operate under the assumption that the information within a report is accurate, report providers (websites) do not guarantee the correctness of their reports.

TABLE I: Examples of inconsistent CTI report claims among sources on the first detection date of Shylock malware.

Source	Part of article
A	Initially discovered in ***February 2011*** by security firm Trusteer, ***Shylock*** delivers web injects into victims' browsers and logs keystrokes.
B	The ***Shylock*** malware, named after a character from Shakespeare's The Merchant of Venice, was first discovered in ***September 2011*** and ...
C	***Shylock*** was discovered in ***September 2011*** and specializes in financial fraud.

Table I presents parts of articles posted on three cybersecurity sources regarding when the Shylock malware was first identified. While source A claims that Shylock was first detected in February 2011, sources B and C assert that this malware was first discovered in September 2011. In this paper, we explore an automated approach to detecting inconsistencies such as this in competing CTI reports.

B. Term Definition

Before the methodology in this paper is presented, let us first define several key terms.

TABLE II: The relation-types and the entity-types that make up the relations.

Relation	Entity	Entity
Exploitation	Malware	CVE
Infection	Malware	Method
Target	Malware	Platform
Detection	Malware	Date

Definition 1 (Entity). An *entity* is something that has its own identity. We consider only the entities classified as one of the five types shown in Table II.

Definition 2 (Relation). A *relation* is an ordered pair of the form (entity, entity), and the relation-type depends on the types of entities constituting the relation, as shown in Table II. A *relation extraction* is the task of extracting the semantic relationships between entities from unstructured text on the Web, more specifically on cybersecurity related websites (we call them *sources*).

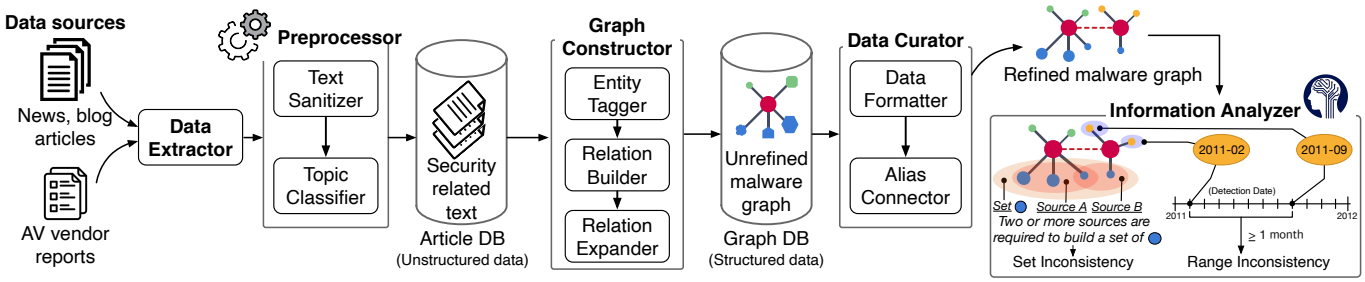


Fig. 1: The architecture of GapFinder.

Definition 3 (Malware graph). A *malware graph* is a graph describing the behavior of malware in terms of the relation-types defined in Table II. For each relation, the two entities become nodes, and the relation between the two entities becomes an edge. A malware graph consists of one common node that refers to malware, and nodes that refer to other types of entities. The edge between the nodes has two attributes: the relation-type, and the source from which a relation is extracted.

III. System Design

Figure 1 illustrates the overall architecture of GapFinder, which consists of five main components: (1) data extractor, (2) preprocessor, (3) graph constructor, (4) data curator, and (5) information analyzer, and two databases for storing different types of data. We first present the overall workflow of GapFinder, followed by a description of each component.

A. Overall Workflow

Let us consider the following real-world example involving the extraction of Dyre malware security relations, and the detection of inconsistencies among these relations. Figure 2 (read from top to bottom) provides a visual representation of the workflow and components. First, the data extractor crawls publicly available security related websites, and inputs the fetched content into the preprocessor, which performs text sanitization and applies a security-topic classification algorithm to validate that the article is relevant to cybersecurity (step 2: Preprocessor). Next, the relevant text is processed by the graph constructor, which derives structured relations from the text for some relation-types. Here, two types of relations, *infection* and *detection*, are extracted. The graph constructor then constructs the malware graphs based on the malware names (first entity of relation), Dyre and Dyreza. (step 3: Graph Constructor). Next, the data formatter normalizes the format of the nodes for the malware graphs (step 4-1: Data Formatter). For example, "June" in the Dyre malware graph is normalized into "2014-06." The alias connector is then applied, linking malware nodes that have different names but appear to indicate the same malware (step 4-2: Alias Connector). Because Dyre malware is also known as Dyreza and Dyzap [25], these different names should be linked to each other (to compare the nodes on different malware graphs). Finally, the information analyzer finds an inconsistency between nodes of the same type in the connected malware graphs (step 5: Information Analyzer). Here, the two date nodes, "2014-06" and "2014-09," make different claims about the detection date of Dyre. According to the inconsistency definition for the detection date of malware, we can infer that there is an inconsistency between these two date nodes.

B. Data Extractor and Preprocessor

1) Data Extractor: The data extractor is essentially a crawler designed to collect articles from a set of cybersecurity websites. For each website, it explores all available links. Table III shows a summary of our collected dataset. The data extractor crawled roughly 474K articles from 119 sources, and the publication dates of the articles ranged from January 2001 to December 2018.

TABLE III: A dataset of unstructured text.

Article time-span	Jan. 2001 ∼ Dec. 2018
# of sources	119
# of articles	474,093
# of sentences	10,516,004

2) Preprocessor: The content of each article scraped in the data extractor includes many noisy characters (e.g., HTML tags, hyperlinks). To extract clean text, a text sanitizer is applied, which isolates the text using a set of regexes.

A second preprocessing step performs topic validation to ensure that articles that enter the dataset are indeed cybersecurity related. Although we choose cybersecurity websites as crawling targets, it is possible that the main topic of a published article is not security. Thus, we designed a topic classifier to sift out non-cybersecurity articles from our data. The topic classifier operates in two phases: (1) word representation, and (2) article classification. First, it trains a Doc2Vec model [20] with the headlines of the collected articles. The Doc2Vec model, which is a word embedding algorithm, generates vector representations of sentences based on the words in the sentences, and places sentences with similar meaning close to each other in a vector space. Because the headline implies the main idea of an article, it is the most appropriate sentence for classifying the article. To obtain a more accurate Doc2Vec model, we removed digits and stop words that had little meaning (e.g., "is," "the") from the headlines. Next, the topic classifier runs a support vector machine (SVM) to classify cybersecurity related pages. Here, to train an SVM model, we used the vector representations

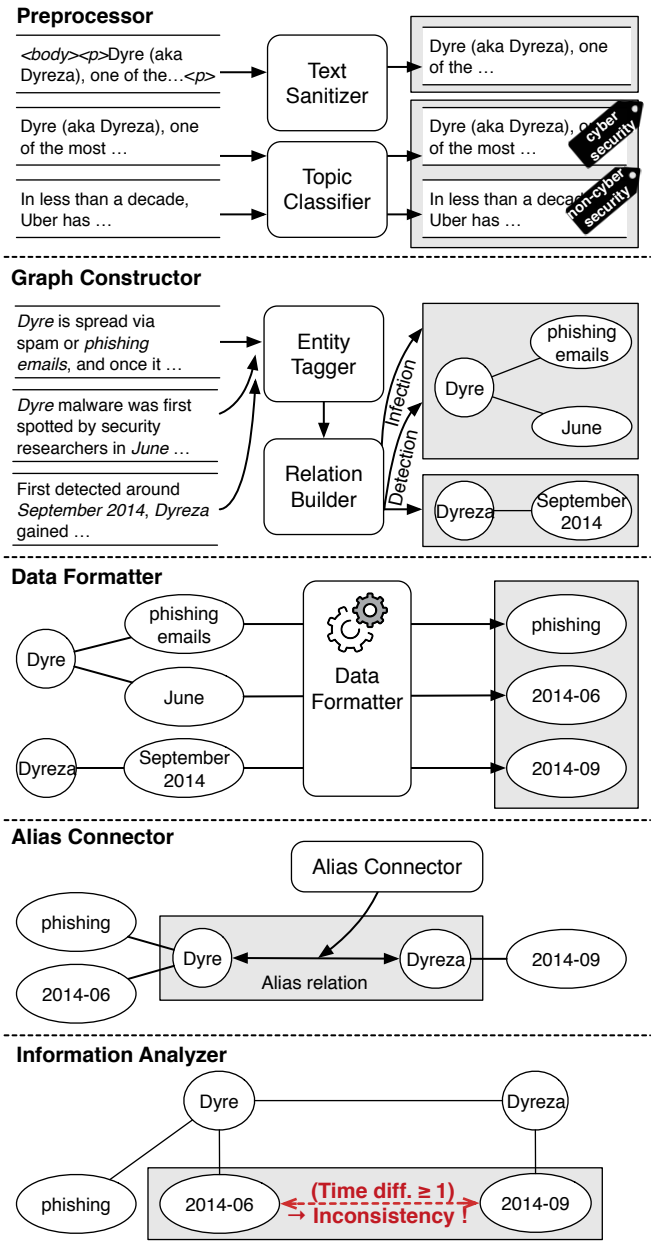


Fig. 2: The overall workflow of GapFinder. The gray boxes correspond to the output of each component.

of 300 `cybersecurity` and 300 `non-cybersecurity` articles as features, and tested the model with another 600 articles (300 articles for each case). The validation showed that the classification provided high precision (94%) and recall (95%). With this approach, we removed approximately 54K non-cybersecurity articles. Finally, the articles classified as the `cybersecurity` are stored in the article DB, and are used as the input for the graph constructor.

C. Graph Constructor

Through the data extractor and the preprocessor, GapFinder collects large volumes of security related sentences. As these extracted sentences are unstructured data, they must be structurally normalized prior to processing. To do this,

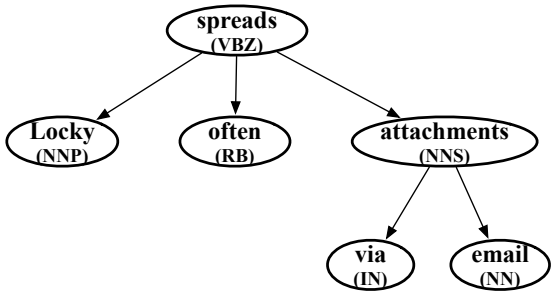


Fig. 3: The output of NLP tasks on the sentence "Locky often spreads via email attachments." The value in parentheses indicates the PoS tag of the word.

one could consider leveraging existing information extraction (IE) techniques [15], [2], [1], [24], [35], [10]. However, these techniques are not ideally suited to addressing the domain specific nature of information that arises in the cybersecurity domain. While there are some approaches [19], [18] that extract security-relevant information using existing IE techniques [10], Liao et al. showed that a direct application of these techniques led to low classification performance in terms of precision (around 70%) and recall (less than 50%) [21]. Thus, we designed the graph constructor, consisting of the entity tagger, relation builder, and relation expander to derive structured data from unstructured text with consideration for security specific issues.

1) The relation-types to be addressed in this work: We focus on extracting structured data about the five most mentioned malware features in the articles. To determine the malware features, we group the verbs, which are placed adjacent to malware names in sentences, into several clusters based on the semantic similarity between the verbs. This approach was derived from the observation that the verbs placed between the subject of action (i.e., malware) and the object of action are mainly used to describe the type of action.

We first construct a malware dictionary, consisting of 398 well-known malware names, based on the Wikipedia pages belonging to the `Malware` category and all its sub-categories. We then perform a dictionary lookup on the sentences stored in the article DB to find sentences containing the malware names. Second, for the filtered sentences, we identify the verbs that are grammatically closest to the malware name through two natural language processing (NLP) tasks: part-of-speech (PoS) tagging and dependency parsing. The PoS tagging determines the PoS of words, and the dependency parsing represents the grammatical structure of sentences. Figure 3 shows the output of NLP tasks on the sentence "Locky often spreads via email attachments." From the malware name "Locky," we move toward the parent word until the word whose PoS tag is the verb (e.g., `VB`, `VBZ`) is first detected. Here, we derived the verb "spreads" from the sentence. Finally, we perform word clustering to group the identified verbs with similar meanings. To be specific, we train a Word2Vec model [23] with the words in the cybersecurity articles. The trained model places the words that share similar contexts and semantic meanings in adjacent positions. We then use the vector representation

of each verb as a feature, and determine K, the number of clusters, using the average silhouette method [27].

TABLE IV: The result of word clustering based on the semantic similarity between the verbs.

Cluster (Topic)	Frequency	Cluster words
Reporting	4,815	said, says, reported
Detection	3,819	found, discovered, seen
Infection	1,973	spread, spreading, distributed
Exploitation	1,349	exploit, exploited, leveraged
Target	1,097	targeted, aimed, affected

Table IV shows the five largest clusters in terms of the frequency of verbs belonging to the same cluster and the top three verbs belonging to each cluster. The topics of these clusters are assigned based on the verbs that belong to each cluster. The largest cluster is related to the `reporting` topic, which includes the verbs that most often appear in quotes. However, it is obviously not related to malware features. On the other hand, the remaining four clusters address malware features: (1) the first detection date of malware, (2) the infection methods used by malware, (3) the CVE identifiers related to malware, and (4) the target platforms of malware. Thus, we focus on extracting structured data on the remaining four clusters.

2) Entity Tagger: Traditional IE systems [34], [26], [31] perform a two-phase process to derive structured data from unstructured text. First, the NER model annotates the entity type of the words in a sentence. Second, the RE model determines the semantic relationships between a pair of entities recognized by the NER model. Likewise, we start by recognizing the entities of interest, defined in Table II, from the sentences stored in the article DB. To do this, we develop a cybersecurity specific entity tagger based on a well-known NER model [10] and also integrate a coreference resolution module tailored for the cybersecurity domain.

The NER model based on the conditional random field (CRF) assigns a sequence of words to one of the following labels: (1) `MW` (malware), (2) `CV` (CVE identifier), (3) `IM` (infection method), (4) `TP` (target platform), (5) `DA` (date), and (6) `O` (others). For example, as shown in Figure 4, given the input sentence, the words "Ramnit" and "emails" are annotated with `MW` and `IM`, respectively, and others are annotated with `O`. Just like any other classifier, the NER model uses several criteria to distinguish the entities of our interest from the other entities. Two simple examples of entity-distinguishing criteria are 1) whether the first letter of the word is capitalized, and 2) what the words are before and after the current word. In cybersecurity articles, the first letter of a malware name is usually capitalized, and a malware name is usually located before the types of malware (e.g., trojan, botnet) or the word "malware."

A generic NER model often misclassifies a malware entity as `O` if it does not consider the malware's pronouns. For example, as shown in Figure 5, after the malware "Dridex" is mentioned at the beginning of the article, the malware type "Trojan" is used as a pronoun to refer to it. However, a generic

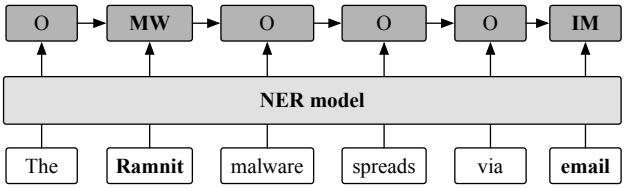


Fig. 4: The NER model that assigns a sequence of words with one of the labels (`MW`, `CV`, `IM`, `TP`, `DA`, and `O`).

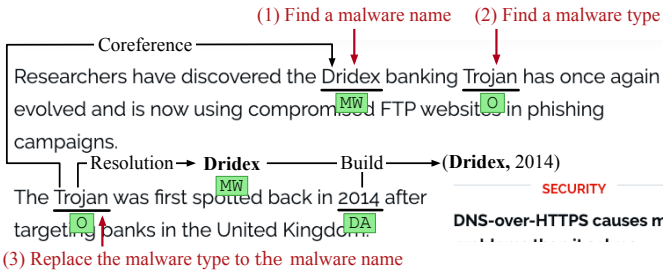


Fig. 5: The process of coreference resolution with a real-world example.

NER model does not recognize "Trojan" as `MW` even though it indicates the malware entity "Dridex." As a result, while "Dridex" is annotated with `MW`, "Trojan" is annotated with `O`. This misclassification prevents the appearance of new entity pairs that may form a `MW-DA` relation.

To address this issue, we design a coreference resolution algorithm that is tailored to the cybersecurity domain. From the beginning of the article, (1) we look for a malware's proper noun, labeled with `MW`, by the NER model. (2) We then look for a noun that indicates a malware type within three words back and forth around the malware name found in (1). Currently, we consider four well-known malware types (i.e., trojan, worm, botnet, and ransomware). If there is no word indicating a malware type, the word "malware" is used by default. From our observation, after the first occurrence of a malware name in an article, the type of malware found in (2) is typically used to refer to the malware. Thus, (3) if the malware type appears after the sentence containing a malware name, it is replaced with the malware name, and the label is also modified from `O` to `MW`. As shown in Figure 5, "Trojan" is replaced with "Dridex," and the label of "Trojan" is modified from `O` to `MW`.

3) Relation Builder: The goal of an RE model is to extract the semantic relationships between the entities recognized by an NER model. However, existing RE models [32], [8] do not verify the labels of entities, which could lead to the extraction of malware-irrelevant relations such as (Huawei, 1987). To address this issue, we introduce a relation builder that performs the verification of entity labels before extracting the structured relations among the entities.

We observed that some words indicating IT companies (e.g., Huawei, OpenDNS) were annotated with `MW`, but they should have been annotated with `O`. To rectify the incorrect labeling, we perform K-means clustering [14] with the vector representations of the entities annotated with `MW`. Our intuition

is that the words indicating malware names and non-malware names are used in different contexts in cybersecurity articles.

TABLE V: The five words closest to "Dyre" and "Huawei" in the Word2Vec model.

Dyre (Distance)	Huawei (Distance)
Dridex (0.90)	ZTE (0.88)
Neverquest (0.86)	Huaweis (0.65)
TrickBot (0.86)	Lucent (0.65)
Zbot (0.85)	Motorola (0.64)
Vawtrak (0.84)	Nokia (0.64)

Table V shows the five words closest to "Dyre" and "Huawei" in the Word2Vec model. While the words related to Dyre are malware names, most of the words related to Huawei are the names of IT companies. Using this feature, we perform K-means clustering to partition the entities annotated with `MW` into two clusters, `malware` and `non-malware`, and modify the labels of the entities that are clustered as `non-malware` with `O`. With this approach, we corrected many erroneous labels of malware irrelevant entities, such as the names of IT companies and the names of hackers.

Then, the RE model assigns the relations between verified entities with one of the following types: (1) exploitation, (2) infection, (3) target, (4) detection, and (5) others. For example, a pair of entities (Ramnit, email) in Figure 4 is annotated with the `Infection` type. The syntactic and lexical feature set for classification is taken from the previous works [24], [34], [26], [32], and some of them are presented below.

- (Syntactic) Lemmas of words in the dependency path.
- (Syntactic) Sequence of dependency labels in the dependency path connecting the heads of the entities.
- (Lexical) PoS tags of the words in a sentence.
- (Lexical) Sequence of words between the entities.

4) Relation Expander: Generally, since multiple malwares are mentioned in a single cybersecurity article, our entity tagger and relation builder focused on extracting relations between entities mentioned in the same sentence. However, AV vendor reports mainly deal with a single piece of malware throughout the report. Thus, the first entity (malware) of the relation defined in Table II is predetermined in a single report. By finding a second entity of interest (one of `CV`, `IM`, `TP`, and `DA`) within a report, we can build relations between the entities even if they do not appear in the same sentence.

Figure 6 shows an example of AV vendor reports [33] and how the relation expander works. A report usually consists of three divisions: (i) title, (ii) structured data, and (iii) unstructured data. The title is the name of malware to be described throughout the report. From this, we extract the first entity of relations. However, most vendors name malwares with detailed malware information, such as target platform (e.g., w32, Linux) and malware type (e.g., trojan, ransom). To obtain only a malware family name, we designed an algorithm based on the observation that there is a somewhat unified order of naming malware among AV vendors: target platform (P) is placed first, followed by malware type (T), family name (F), and variant version (V) (the details are shown in Appendix). We next derive the second entities from

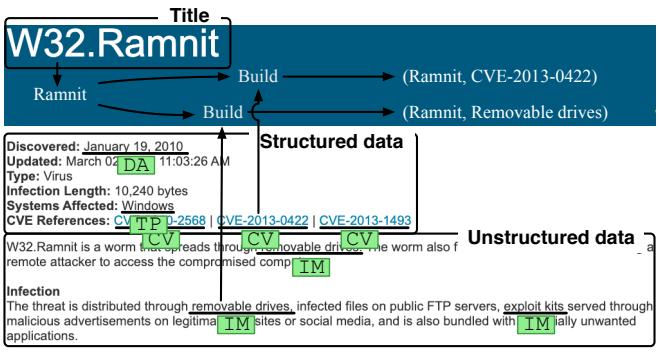


Fig. 6: An example of AV vendor reports, and the process of relation expansion.

structured and unstructured data. Since the structured data is composed of a *property:value* pair, we directly read the values of the property and annotate them with proper entity-types. For unstructured data, the entity tagger annotates all words in sentences with one of the entity-types. Finally, we build relations between the malware name and the words annotated with one of the entity-types (`CV`, `IM`, `TP`, and `DA`). For example, an exploitation relation (Ramnit, CVE-2013-0422) would be formed between "Ramnit" in the title and "CVE-2013-0422" annotated with `CV`.

5) Malware Graph Generation: For the extracted relations from the relation builder and relation expander, the graph constructor builds the malware graphs based on the relations of our interest and stores them into the graph DB, which is used as the input for the data curator. For each relation, two entities become nodes, and the relation between the two entities becomes an edge in the graph. If there is more than one relation consisting of the same entity pair, multiple edges are created between a single pair of nodes.

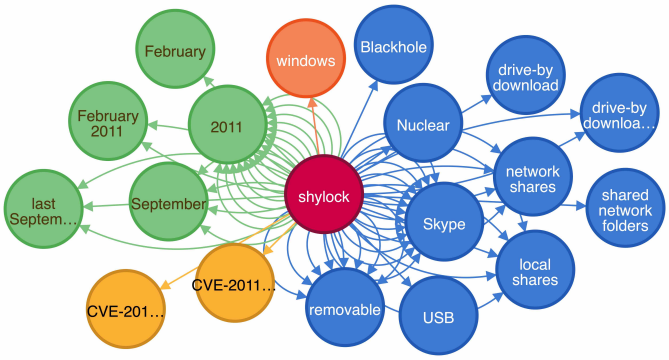


Fig. 7: Unrefined Shylock malware graph.

Figure 7 shows an example of malware graphs. The node color indicates the type of entity: red for malware, blue for infection method, orange for target platform, yellow for CVE identifier, and green for date. The attributes of the edges are omitted to improve the visibility of the graph.

D. Data Curator

There are two challenges to be tackled before finding data inconsistency in malware graphs. First, while the data

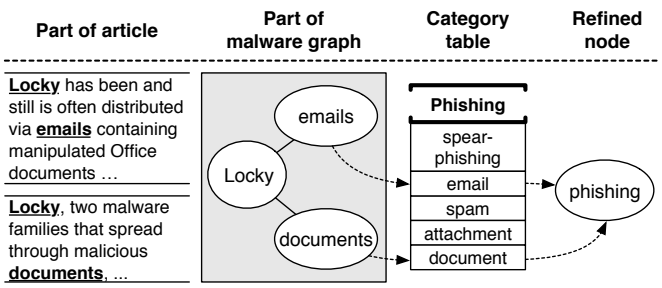


Fig. 8: The data refinement process to make different words with a similar meaning into the same representation.

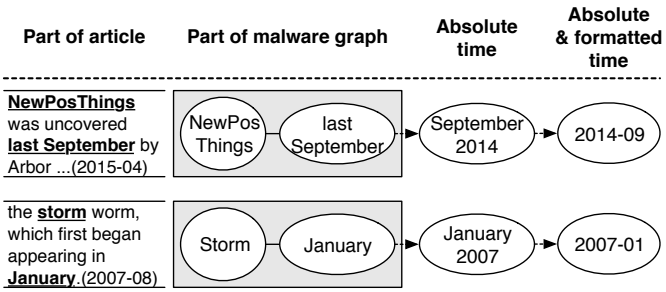


Fig. 9: The two-phase data refinement process to obtain absolute and formatted date nodes. The date in parentheses of article indicates the date of writing.

used in existing truth discovery studies [38], [39] follow a specific format, the nodes of the malware graphs are raw data that require a form of normalization. Some node types have different shapes to indicate similar or equivalent meanings. Therefore, these semantically similar nodes should be refined with the same representation. Second, since malware graphs are disconnected from each other, the nodes on different graphs cannot be compared with each other. However, some malwares are referred to by a number of different names, which are independently derived by AV vendors. Therefore, an additional process is required to compare nodes from different malware graphs that refer to the same malware. We now present a data curation process to address these challenges.

1) Data Formatter: The data formatter performs data refinement to create common expressions for all nodes with similar or identical meanings. From our observation, there are two types of nodes that need to be refined: (i) infection method, and (ii) date. Since the CVE identifier has a standard format, and the target platform is one of a finite number of operating systems, no additional data refinement process is required.

Figure 8 shows a part of the Locky malware graph, consisting of two infection relations, (Locky, emails) and (Locky, documents). While the two infection method nodes are different words, in a broad sense, they mean the same infection technique "phishing," shown in the upper left sentence of Figure 8. Therefore, we need to refine different words with a similar contextual meaning into a single representation.

To do this, we first classify infection methods into six categories: (1) phishing, (2) network, (3) removable (device), (4) exploit (kit), (5) social, and (6) drive-by (download). We then build the category tables with the five words closest to each category in the Word2Vec model. For each infection method node stored in the graph DB, if it corresponds to one of the words in a category, it is replaced with the category to which the word belongs. To consider the case where the node is used in plural form, we normalize the node into its base form using lemmatization. From the data refinement, both the words "emails" and "documents" belonging to the `Phishing` category are transformed into the same word "Phishing," shown in Figure 8.

Second, there are various ways to express time information. For example, time could be expressed with a relative (e.g., last September) or absolute expression (e.g., September 2014). The year could also be omitted, or the month could be abbreviated (e.g., January to Jan). Therefore, to compare the date nodes of the malware graphs, we need to make unrefined date nodes into absolute and formatted time expressions.

Figure 9 shows a two-phase process to obtain absolute and formatted date nodes. First, the data formatter computes the absolute times of unrefined date nodes (we call them input) through the time expressions contained in the inputs. For example, the word "last" in the input "last September" indicates the most recent past "September" from the date of writing. Here, since the date of writing is April 2015, "last September" should be September 2014. If the date node does not contain the year information, we consider that the year of the node is the same as the year of the date of writing. For example, as shown in the Figure 9, the input "January" is transformed into "January 2007" with reference to the year of the date of writing (2007-08). Second, the data formatter modifies the absolute time expression into a formatted one (i.e., yyyy-mm) using the predefined rule set [4]. Finally, we get the absolute and formatted date nodes, "2014-09" and "2007-01," from "last September" and "January," respectively.

2) Alias Connector: While most entities around us are distinguished by a unique name (e.g., company, person), malwares commonly have many aliases due to the different naming conventions of AV vendors. This feature prevents the nodes related to the same malware from being compared to each other, since malware graphs are separated based on malware names. For example, according to the Conficker Wikipedia page, Downadup is another name for Conficker. However, as shown in the bottom of Figure 10, the nodes in the two graphs cannot be compared with each other (i.e., 2008-10 vs., 2008-11). To address this problem, we need to connect the malware graphs with different names indicating the same malware.

Figure 10 shows the procedure for linking different malware graphs. (1) We first extract (malware, malware) pairs that have alias relations from the Web. Here, we use the malware dictionary constructed from malware related Wikipedia pages (see in Section III-C) as the data source. Since the pages mainly consist of two data structures: (i) unstructured data (i.e., body section), and (ii) structured data (i.e., infobox), we apply different methods to extract alias relations depending on the data structure. For the unstructured data, we leverage

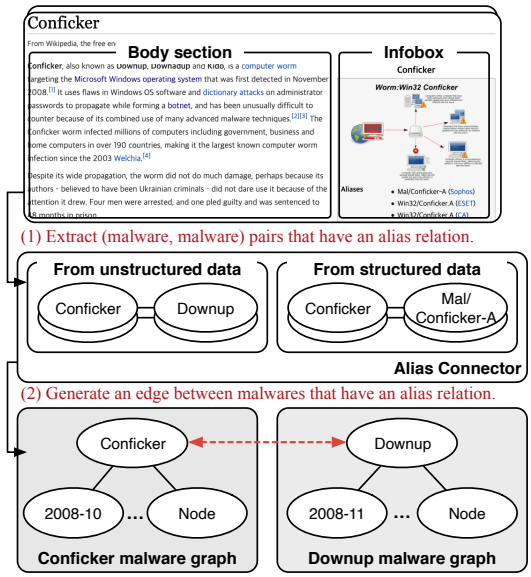


Fig. 10: The procedure for linking different malware graphs.

our graph constructor since the entity tagger could recognize the malware name in unstructured text. Moreover, most alias relations are represented with some limited phrases such as "also known as", "called", and "named." This distinctive feature enables the relation builder to extract alias relations more effectively. For example, an alias relation (Conficker, Downup) is extracted from the part of the sentence "Conficker, also known as Downup, ..." on the Conficker page. For the structured data, alias relations are obtained by mapping the title of a Wikipedia page (i.e., malware name) into the values corresponding to "Aliases" in the infobox of the page. For each alias relation, (2) the alias connector generates an edge between two malware entities that have an alias relation (red dotted line in Figure 10).

E. Information Analyzer

After refining the nodes and linking between malware aliases, the information analyzer detects whether data inconsistency exists or not for each connected malware graph. If there is only one correct and definite answer to a question, we can easily define an inconsistency: If two answers are simply different for the same question, there is an inconsistency between these two answers. For example, consider the following question: "Where is the capital of the United States?" If one receives two different answers, for example, Washington DC and New York, detecting the inconsistency is straightforward. However, this simple comparison cannot be directly applied to the cybersecurity domain for two reasons: (i) multiple truths, and (ii) the uncertainty of truth.

First, most malwares use two or more infection methods to compromise benign hosts and target two or more operating systems. For example, Conficker spreads via removable media and network shares [9]. Therefore, it is difficult to determine if there is an inconsistency even though each node is different. Second, since AV vendors conduct independently analyses and do not share their findings with each other, the first

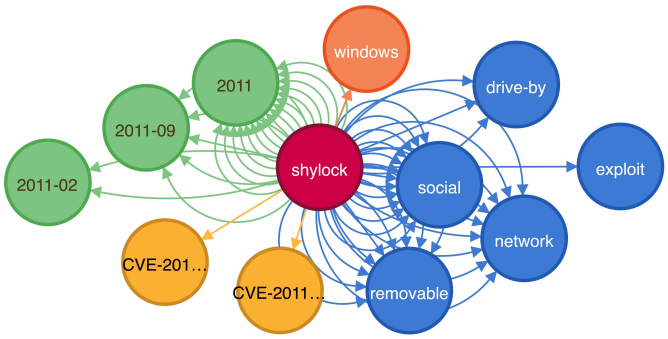


Fig. 11: Refined Shylock malware graph.

detection dates of a malware outbreak may vary from vendor to vendor. Therefore, there is no definite correct answer for the detection date. To address these matters, we introduce two types of inconsistencies: (i) set inconsistency for exploitation, infection, and target relation and (ii) range inconsistency for detection relation with the example malware graph in Figure 11, which has completed the data curator step.

Set inconsistency refers to a situation in which two or more sources are needed for the construction of the base set of a specific edge-type. First, we build the base sets for each edge-type. Our work defines the union of nodes belonging to a specific edge-type as the base set for that type. For example, in Figure 11, the base set of infection-type is composed of five nodes: removable, network, social, drive-by, and exploit. For each edge-type, we then remove the nodes covered by the source that accounts for the largest proportion of the base set and check if there are any remaining nodes in the base set. If the base set is not empty, it means that at least two sources are required to construct the base set. In the example above, Threatpost, which accounts for the largest proportion of the base set for infection-type, covered four infection methods. However, the fact that Shylock malware uses an exploit kit is only mentioned by SecurityWeek. Therefore, we consider that there is a set inconsistency in the Shylock malware graph.

Range inconsistency refers to a situation where there are one or more date nodes with a time difference of at least one month from a base date (we call them inconsistent dates). First, we select the base date from all date nodes. Our work defines the date node with the highest degree as the base date. However, a base date that contains only year information (e.g., 2011) makes it difficult to find out the inconsistency between the nodes that have month information (e.g., 2011-02 vs., 2011-09). Therefore, we also consider whether the date node contains month information when selecting the base date. As a result, the "2011-09" node becomes the base date in Figure 11. We then check whether each date node has a time difference of more than a month from the base date. In the example above, since the time difference between the "2011-02" node and the base date is seven months, we consider that there is a range inconsistency in the Shylock malware graph.

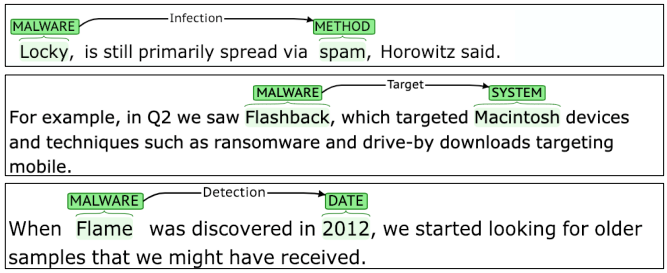


Fig. 12: Examples of annotated sentences.

IV. Evaluation

A. Dataset

To train the graph constructor, we first construct a ground truth dataset by manually annotating the entities and their relations defined in Section II-B. To do this, for each relation-type, we randomly extract $1,000$ sentences that contain the verbs presented in Table IV from the article DB, which is likely to contain relations within the sentences. For each sentence, we employed five computer science graduate students to manually annotate the sentences based on references made to the following entity types: 1) describes a malware, 2) references a CVE, 3) discloses a method, 4) identifies a platform, and 4) references a date. In addition, the students were asked to produce relation connections, as shown in Figure 12. Disagreements in these manually produced annotations were subsequently resolved through a majority vote.

Table VI shows the training dataset (i.e., ground truth) used for training the graph constructor and the dataset extracted from the graph constructor.

TABLE VI: Datasets of structured relations.

Training set		**Extracted set**	
Relation-type	**# of relations**	**Relation-type**	**# of relations**
Exploitation	158	Exploitation	2,894
Infection	663	Infection	6,390
Target	186	Target	13,933
Detection	789	Detection	23,204
Total	**1,796**	**Total**	**46,421**

B. Landscape

GapFinder extracted four types of structured relations from unstructured text, and constructed malware graphs for each malware. Table VII presents the five largest malware graphs in terms of the degrees of malware nodes. In general, we observed that the greater the impact of malware on individuals and communities (either by its spread or its potential harm), the greater the number of articles about the malware found by our crawler. In addition, the larger the number of articles found for a given malware, the likelier it was that structured relations on the malware would be derived. Therefore, the scale of a malware graph seems predictive to the potential social impact of that malware. For example, Stuxnet, which targeted industrial control systems, and Conficker, had a great impact on society, were ranked in the top five malware graphs.

TABLE VII: The five largest malware graphs in terms of the degrees of malware nodes.

Malware graph	Stuxnet	Conficker	Zeus	Locky	WannaCry
# of degrees	510	314	310	210	147

Table VIII shows the top five nodes with the highest degree for three types of nodes. Among the CVE-type nodes, CVE-2012-0158 is involved with the largest number of malwares (e.g., Zbot, NetTraveler). In addition, all CVE identifiers have high CVSS scores above 9.0. The common vulnerability scoring system (CVSS) [11] measures the severity of a CVE identifier with a score between 0.0 and 10.0 (higher is more critical). Among the method-type nodes, *phishing* was used by the largest number of malwares. In fact, *email* have the largest proportion of infection methods mapped to *phishing* (by the data formatter), followed by *spam*. Finally, the Windows family occupied the top five positions of the target platforms. The word "Windows" accounts for the largest proportion (40.6%), meaning that most articles do not precisely identify the susceptible target platforms down to their specific OS versions.

TABLE VIII: The top five nodes with the highest degree for three types of nodes.

CVE	**% of nodes**	**Method**	**% of nodes**	**Platform**	**% of nodes**
2012-0158	4.2	Phishing	51.3	Windows	40.6
2012-0507	2.1	Social	13.8	2000	13.9
2011-3544	2.0	Exploit	11.1	Server 2003	13.5
2012-0507	1.9	Removable	9.0	XP	11.6
2017-0199	1.9	Network	7.4	7	3.9

C. Understanding Inconsistency

GapFinder found several data inconsistencies among the malware graphs. Here, we describe the forms of data inconsistency that were illuminated through this process.

1) Set inconsistency: Figure 16 illustrates the heatmap of Jaccard similarity between the base sets of the ten malware samples and the infection methods of these malware samples mentioned in the ten sources. Jaccard similarity is used to measure the similarity of two sets, where the similarity is in a range between 0 and 1. The value placed at (x, y) in the heatmap indicates how much the infection method set on the y-axis source covers the base set of the x-axis malware. The higher the value, the higher the coverage. GapFinder identified set inconsistencies in the nine malware samples, except for Koobface. No source covered the base set of these malware families single-handedly, and at least two or more sources are required to construct the base set of the malware samples. For example, the base set of Locky ransomware consists of five infection methods, and at least three sources (i.e., securityweek, scmagazine, and zdnet) are required to fully describe Locky's base functionality. Alternatively, the site, The Register, covers the base set of Koobface functionality (1.00 in Koobface row and theregister column in Figure 16).

Figure 13 and 14 show the coverage of the top three sources dealing with the base set of malware samples in terms of

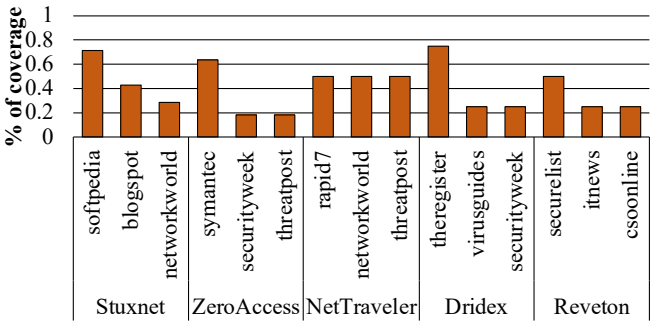


Fig. 13: The coverage of sources on the base set of given malware samples in terms of exploitation relation.

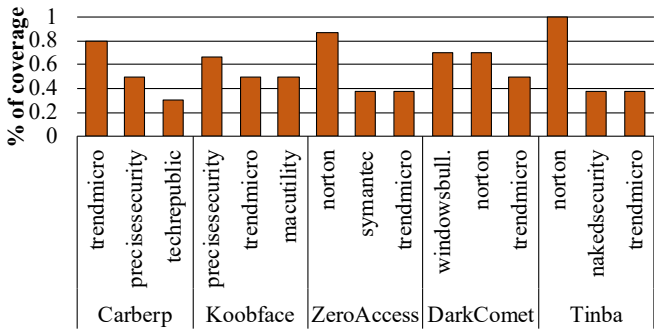


Fig. 14: The coverage of sources on the base set of given malware samples in terms of target relation.

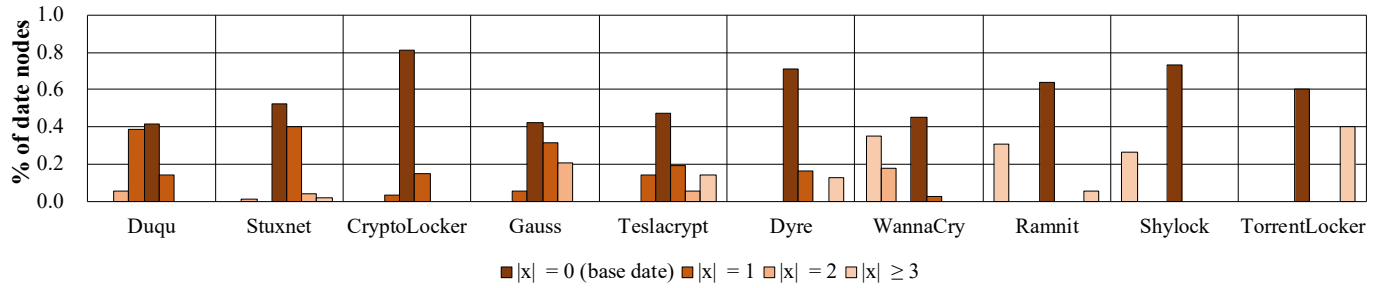


Fig. 15: The distribution of date nodes for ten malware graphs. The x in the legend is the time difference between a base date and inconsistent date nodes.

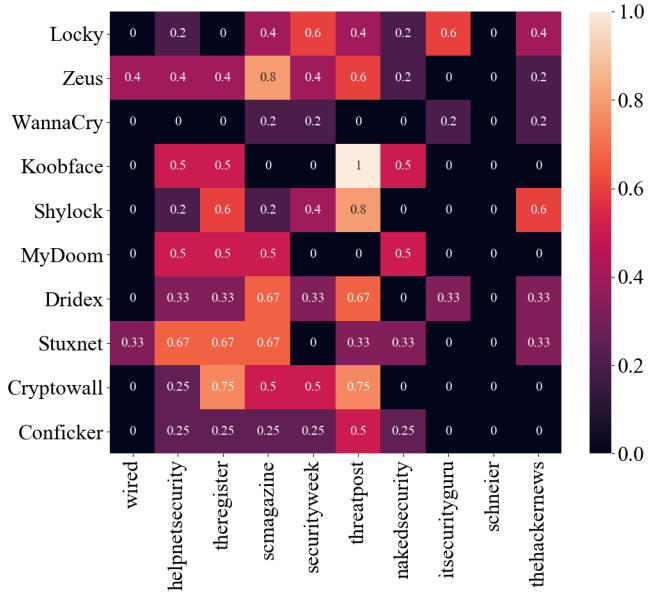


Fig. 16: The heatmap of Jaccard similarity between the base sets of the ten malware samples and the infection methods of these malware samples mentioned in the ten sources.

exploitation and target relations. No source mentions all CVEs related to these malware samples. Even the source covering the largest proportion of the base set mentions only 62.0% of the base set on average. For example, ZeroAccess needs exactly three unique sources to cover the base set. The CVEs that each source refers to are not mentioned in any other sources. In the case of target relation, on average, the sources covering the largest proportion of the base set addressed more than 80.1% of the base set.

2) Range inconsistency: Figure 15 shows the distribution of date nodes, including a base date and inconsistent date nodes, for ten malware graphs where range inconsistencies were found. For the first six malware samples, the inconsistent dates, which differ by one month from the base date ($|x| = 1$), account for the largest proportion of the total inconsistent dates. On the other hand, for the last four malware samples, the inconsistent dates are far from the base date ($|x| = 2$ or $|x| \geq 3$). For example, in the case of Ramnit malware, while the base date is April 2010, two different inconsistent dates are January 2010 and July 2010. They were extracted from the sites Symantec and PCingredient, respectively. The malware graph with the longest time difference between the base date and inconsistent dates was Shylock malware: February 2011 versus September 2011.

D. Case Study

In this subsection, we perform case studies on the data inconsistencies that GapFinder found.

1) Set inconsistency:

Use Case 1. (Inconsistency in CVE identifiers). Figure 17 shows a part of Stuxnet malware graph consisting of exploitation relations. Through the multiple edges between the nodes "Stuxnet" and "CVE-2010-2568," we determined

TABLE IX: The three use cases related to range inconsistency.

Use case	Malware	Date	Example sentence containing (malware, date) relation
1	CryptoLocker	2013-08	**[Cryptolocker]** is a rather unpleasant strain of malware, first spotted in **[August]**, that ...
		2013-09	**[CryptoLocker]** first surfaced in **[September 2013]**, with P2P ZeuS (aka Gameover ZeuS) ...
2	Shylock	2011-02	Initially discovered in **[February 2011]** by security firm Trusteer, **[Shylock]** delivers web ...
		2011-09	**[Shylock]**, a threat first observed by Trusteer in **[September 2011]**, was named after a ...
3	Dyre	2014-06	**[Dyre]**, also known as Dyreza, is a banking Trojan that was first seen around **[June 2014]**.
	Dyreza	2014-07	**[Dyreza]**, which first appeared in **[July 2014]**, is also capable of exploiting Chrome, ...
		2014-09	First detected around **[September 2014]**, **[Dyreza]** gained notoriety for its ability to bypass ...

that various sources mentioned that Stuxnet leverages CVE-2010-2568. However, Stuxnet actually uses a large number of CVEs, which are only extracted from a few sources. For example, the fact that Stuxnet uses CVE-2008-4250, which is also used in Conficker botnet, was only collected from one source. This kind of inconsistency between sources makes it difficult to analyze the relationship between malwares and CVEs. For example, if we had not collected the articles of the *marcoramilli* (a blog source), we would not have noticed that Conficker, which appeared in November 2008, and Stuxnet, which appeared in June 2010, employed the use of a common CVE vector.

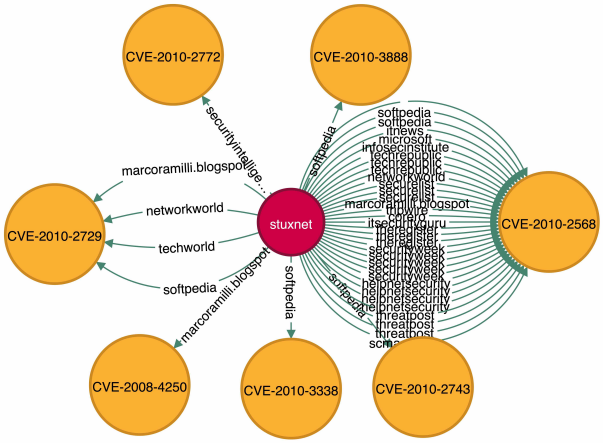


Fig. 17: A part of Stuxnet malware graph consisting of exploitation relations.

Use Case 2. (Inconsistency in infection methods). The Shylock malware spreads itself via five infection methods. While three infection methods are mentioned in many sources, drive-by downloads were addressed in two sources, and infection via exploit kit was addressed in one source. We observed that, like other aspects of malware behavior, the depth and breadth of this information varies from source to source. Therefore, when analyzing malware behavior, we should consider multiple cybersecurity sources to offset the information that some sources do not cover.

2) Range inconsistency:

Use Case 1. (Inconsistency in the same source). Most range inconsistencies arise between dates mentioned in different sources. However, even within articles from the same publisher (e.g., from The Register), we found different claims (i.e., 2013-08 vs., 2013-09) regarding the detection date of Cryptolocker, shown in use case 1 of Table IX. Most previous studies on truth discovery [36], [12], [6] find trustworthy information using the following approach: (i) the history of a source in producing reliable information results in an increasing reputation metric for judging credibility, (ii) the larger the number of credible sources that repeat an assertion, the greater the probability is that the assertion is accurate. However, the presence of inconsistency in the same source makes it difficult to apply existing truth discovery approaches to the cybersecurity domain, which could be an open challenge.

Use Case 2. (Inconsistency in citation to the same AV vendor). The use case 2 of Table IX shows the excerpts from two articles citing the comments of AV vendors to describe the first detection date of Shylock malware. The problem is that two excerpts cite the same AV vendor (Trusteer), but the time difference between the dates they claim is seven months (i.e., 2011-02 vs., 2011-09). According to Trusteer, the object of the citation, [29], Shylock malware was first discovered in September 2011, which is also the base date of the malware. Then, the first excerpt misquotes the AV vendor, Trusteer. This also shows that trustworthy information would be better found in more specialized sources in the cybersecurity domain.

Use Case 3. (Inconsistency between malware aliases). The use case 3 of Table IX shows why the alias connector is needed to find data inconsistency. From the connected malware graph between Dyre and Dyreza, we found three different claims about the detection date of the malware (i.e., 2014-06 vs., 2014-07 vs., 2014-09). In addition to the malware names, there are numerous terms indicating the same meaning in the cybersecurity domain. For example, there are abbreviations such as EK for exploit kit, IE for internet explorer. Therefore, a cybersecurity specific dictionary is required to scale up malware graphs to more general cybersecurity graphs.

V. Discussion

As we designed our inconsistency checking system, we addressed many language-specific issues to improve its overall accuracy. However, areas for further improvement remain. First, our focus has been on identifying data inconsistency within malware-focused information sources. However, as shown in prior work [8], there arise inconsistencies between versions for the same vulnerable software name. There may be inconsistencies within the various contexts of the cybersecurity related information. In future work, we will seek to expand the scope of relation-types.

Second, we tentatively defined the base date and set as truth. However, the union of nodes (i.e., base set) and the

node with the largest number of degrees (i.e., base date) may not be correct. If a malicious source indiscriminately posts misinformation, the answer may be different. To address this issue, there are some ways to determine the truth from nodes, such as through a reputation-based weighting scheme. Future work will need to consider this issue.

Finally, since existing truth discovery studies are based on structured or formatted data, the data itself contains no noise. However, since our work found data inconsistencies existing in unstructured text, it is inevitable to avoid noises generated during relation extraction. Therefore, while we filtered out noisy relations to a substantial degree, there remains a need to refine the extracted relations.

VI. Related Work

A. Cyber threat intelligence (CTI) with open information

Recently, researchers have proposed ways of automatically finding/understanding cyber threats with publicly available information, such as blog articles and CVE descriptions. Sabottke et al. proposed the mining of tweets related to cyber attacks as a means of designing a real-world early warning system for identifying new exploits [28], and Zhu et al. presented FeatureSmith that generates feature sets for detecting Android malware via text-mining natural language reports [40]. Liao et al. [21] and Catakoglu et al. [3] proposed an automated system for extracting IOCs (e.g., IPs, MD5 hashes) from unstructured text sources. Husari et al. presented an automated, context-aware analysis system for CTI [16]. Our work is orthogonal to the previous works, since our system checks the inconsistency of public information before the previous works use the information. As seen in Section IV, there are many inconsistencies among publicly available information. Our work could improve on the previous works in obtaining more trustable sources.

B. Relation extraction for cybersecurity

There have been some works that extracted cybersecurity relations from unstructured text [19], [22], [18]. Joshi et al. presented an automatic framework that generates relations between cybersecurity concepts and vulnerability descriptions based on supervised learning [19]. McNeil et al. [22] and Jones et al. [18] applied a bootstrapping method to derive cybersecurity relations. While the previous works showed the possibility of relation extraction in the cybersecurity domain, they did not consider the relation extraction in a large-scale environment. Generally, a data correction step, either by manual [30] or semi-automatic methods [5], is required to rectify the misclassified entities and relations in a large-scale environment. Our work derived tens of thousands of relations from 470K articles and proposed a methodology for correcting the misclassified entities and relations.

C. Fact checking in information retrieval

Finding correct information has been also studied in other domains, specifically in information retrieval. Several recent studies have tried to determine correct information from inconsistent information [36], [38], [37]. Yin et al. presented TruthFinder, a general framework for fact validation, that utilizes the relationships between websites and their information [36], and Zhao et al. proposed a new truth-finding method designed to address numerical data [38]. While they also focused on the truth-finding problem, they only consider structured data and did not include unstructured texts. Thus, those approaches cannot be used to find inconsistencies in open web information. Zhang et al. proposed a method for extracting trustworthy information from unstructured data [37]. However, their focus was on a special case of unstructured text for answering specific questions, rather than handling general unstructured texts. Dong et al. first presented an approach to identifying inconsistencies in cybersecurity related information, specifically vulnerable software names and vulnerable versions [8]. However, they targeted only one relation-type, and dealt with entities that were proper nouns or followed a somewhat fixed format, and thus their approach was quite limited. Consequently, the previous work could not compare the entities composed of unstructured text, which makes it impossible to find inconsistencies between the entities. In contrast, our work provides an inconsistency checking system that could handle entities composed of unstructured text by addressing a language specific issue (i.e., similar meaning but different words) and a malware domain specific issue (i.e., malware aliases).

VII. Conclusion

Most CTI projects focus on how to extract and analyze open source information for categorizing and labeling information, as well as deriving inferences that extend threat intelligence beyond the collected elements. However, far less attention has been invested in techniques to ensure the consistency and accuracy of the raw information extracted from the source material. In this paper, we present a novel inconsistency checking system, GapFinder, which is capable of analyzing the structured relations that are extracted from a range of free-form cybersecurity sources. Using a large set of text-mined malware reports from malware-focused sites, our work finds that there exist a range of syntactic and semantic inconsistencies that hinder real-world CTI production. These inconsistencies result not just from the natural difference among groups, but from their reporting nomenclature, level of detail, and factual understanding. Indeed, simply the sources chosen and the length of the data collection can result in a different factual understanding of a given malware. We believe that future CTI studies will benefit significantly from systems such as GapFinder, as it automates data reliability verification.

Appendix

Here, we describe a heuristic algorithm for identifying a malware family name, $family$, when a malware full name, $name$, is given, as shown in Algorithm 1. The algorithm first initializes dictionary variables that are used for looking up the malware properties (lines 1 to 4). Next, the input name is tokenized by the delimiters '.', '\', and '-' (line 5), which

Algorithm 1: The identification of malware family name.

Input : Malware full name, $name$
Output: Malware family name, $family$

```
1  platforms ← [w32, win32, linux, android, osx, ...]
2  types ← [troj, ransom, backdoor, worm, ...]
3  variants ← [A-z],
4  malware ← [mirai, shylock, conficker, ...]
5  tokens ← Tokenize(name,'.|/|-')
6  if tokens.length = 2 then
7      if tokens[0] ∈ types then
8          family ← tokens[1]                    // (T/F)
9      else if tokens[0] ∈ platforms then
10         family ← tokens[1]                    // (P/F)
11     else if tokens[1] ∈ variants then
12         family ← tokens[0]                    // (F/V)
13     else if tokens[0], tokens[1] ∈ malware then
14         family ← tokens[0]                    // (F/F)
15 else if tokens.length = 3 then
16     if tokens[0] ∈ types then
17         if tokens[2] ∈ variants then
18             family ← tokens[1]                // (T/F/V)
19     else if tokens[0] ∈ platforms then
20         if tokens[2] ∈ variants then
21             family ← tokens[1]                // (P/F/V)
22 if family = ∅ then
23     family ← name
```

are used to add malware properties such as target platform, and type in malware names. Then, we derive a malware family name from the input name based on the following observation. AV vendors follow a somewhat unified order of naming malware: target platform (P) is placed first, followed by malware type (T), family name (F), and variant version (V). And, there are some patterns in the malware full name depending on the number of tokens. For example, when the first token is one of the malware types, a family name is commonly located in the second token (lines 7 to 8).

ACKNOWLEDGMENT

This research was supported by the Engineering Research Center Program through the National Research Foundation of Korea (NRF) funded by the Korean Government MSIT (NRF-2018R1A5A1059921).

REFERENCES

[1] E. Agichtein and L. Gravano. Snowball: Extracting relations from large plain-text collections. In *Proceedings of the fifth ACM conference on Digital libraries*, pages 85–94. ACM, 2000.

[2] S. Brin. Extracting patterns and relations from the world wide web. In *International Workshop on The World Wide Web and Databases*, pages 172–183. Springer, 1998.

[3] O. Catakoglu, M. Balduzzi, and D. Balzarotti. Automatic extraction of indicators of compromise for web applications. In *Proceedings of the 25th International Conference on World Wide Web*, pages 333–343. International World Wide Web Conferences Steering Committee, 2016.

[4] A. X. Chang and C. D. Manning. Sutime: A library for recognizing and normalizing time expressions. In *Lrec*, volume 2012, pages 3735–3740, 2012.

[5] O. Deshpande, D. S. Lamba, M. Tourn, S. Das, S. Subramaniam, A. Rajaraman, V. Harinarayan, and A. Doan. Building, maintaining, and using knowledge bases: a report from the trenches. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 1209–1220, 2013.

[6] X. L. Dong, L. Berti-Equille, and D. Srivastava. Integrating conflicting data: the role of source dependence. *Proceedings of the VLDB Endowment*, 2(1):550–561, 2009.

[7] X. L. Dong, E. Gabrilovich, K. Murphy, V. Dang, W. Horn, C. Lugaresi, S. Sun, and W. Zhang. Knowledge-based trust: Estimating the trustworthiness of web sources. *Proceedings of the VLDB Endowment*, 8(9):938–949, 2015.

[8] Y. Dong, W. Guo, Y. Chen, X. Xing, Y. Zhang, and G. Wang. Towards the detection of inconsistencies in public security vulnerability reports. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*, pages 869–885, 2019.

[9] F-Secure. Conficker. https://blog.f-secure.com/what-weve-learned-from-10-years-of-the-conficker-mystery/.

[10] J. R. Finkel, T. Grenager, and C. Manning. Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of the 43rd annual meeting on association for computational linguistics*, pages 363–370. Association for Computational Linguistics, 2005.

[11] FIRST. Common vulnerability scoring system. https://www.first.org/cvss/.

[12] A. Galland, S. Abiteboul, A. Marian, and P. Senellart. Corroborating information from disagreeing views. In *Proceedings of the third ACM international conference on Web search and data mining*, pages 131–140. ACM, 2010.

[13] A. Gandomi and M. Haider. Beyond the hype: Big data concepts, methods, and analytics. *International journal of information management*, 35(2):137–144, 2015.

[14] J. A. Hartigan and M. A. Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, 1979.

[15] M. A. Hearst. Automatic acquisition of hyponyms from large text corpora. In *Proceedings of the 14th conference on Computational linguistics-Volume 2*, pages 539–545. Association for Computational Linguistics, 1992.

[16] G. Husari, E. Al-Shaer, M. Ahmed, B. Chu, and X. Niu. 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. ACM, 2017.

[17] E. M. Hutchins, M. J. Cloppert, and R. M. Amin. Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains. *Leading Issues in Information Warfare & Security Research*, 1(1):80, 2011.

[18] C. L. Jones, R. A. Bridges, K. M. Huffer, and J. R. Goodall. Towards a relation extraction framework for cyber-security concepts. In *Proceedings of the 10th Annual Cyber and Information Security Research Conference*, page 11. ACM, 2015.

[19] A. Joshi, R. Lal, T. Finin, and A. Joshi. Extracting cybersecurity related linked data from text. In *Semantic Computing (ICSC), 2013 IEEE Seventh International Conference on*, pages 252–259. IEEE, 2013.

[20] Q. Le and T. Mikolov. Distributed representations of sentences and documents. In *International Conference on Machine Learning*, pages 1188–1196, 2014.

[21] X. Liao, K. Yuan, X. Wang, Z. Li, L. Xing, and R. Beyah. 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. ACM, 2016.

[22] N. McNeil, R. A. Bridges, M. D. Iannacone, B. Czejdo, N. Perez, and J. R. Goodall. Pace: Pattern accurate computationally efficient bootstrapping for timely discovery of cyber-security concepts. In *2013 12th International Conference on Machine Learning and Applications*, volume 2, pages 60–65. IEEE, 2013.

[23] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[24] M. Mintz, S. Bills, R. Snow, and D. Jurafsky. Distant supervision for relation extraction without labeled data. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2-Volume 2*, pages 1003–1011. Association for Computational Linguistics, 2009.

[25] NJCCIC. Dyre. https://www.cyber.nj.gov/threat-profiles/trojan-variants/dyre.

[26] D. Roth and W.-t. Yih. Global inference for entity and relation identification via a linear programming formulation. *Introduction to statistical relational learning*, pages 553–580, 2007.

[27] P. J. Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65, 1987.

[28] C. Sabottke, O. Suciu, and T. Dumitraș. Vulnerability disclosure in the age of social media: exploiting twitter for predicting real-world exploits. In *24th {USENIX} Security Symposium ({USENIX} Security 15)*, pages 1041–1056, 2015.

[29] SecurityIntelligence. Merchant of fraud returns: Shylock polymorphic financial malware infections on the rise. https://securityintelligence.com/merchant-of-fraud-returns-shylock-polymorphic-financial-malware-infections-on-the-rise/.

[30] J. Shin, S. Wu, F. Wang, C. De Sa, C. Zhang, and C. Ré. Incremental knowledge base construction using deepdive. In *Proceedings of the VLDB Endowment International Conference on Very Large Data Bases*, volume 8, page 1310. NIH Public Access, 2015.

[31] M. Surdeanu and M. Ciaramita. Robust information extraction with perceptrons. In *Proceedings of the NIST 2007 Automatic Content Extraction Workshop (ACE07)*, 2007.

[32] M. Surdeanu, D. McClosky, M. R. Smith, A. Gusev, and C. D. Manning. Customizing an information extraction system to a new domain. In *Proceedings of the ACL 2011 Workshop on Relational Models of Semantics*, pages 2–10. Association for Computational Linguistics, 2011.

[33] Symantec. Ramnit. https://www.symantec.com/security-center/writeup/2010-011922-2056-99.

[34] L. Yao, S. Riedel, and A. McCallum. Collective cross-document relation extraction without labelled data. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 1013–1023. Association for Computational Linguistics, 2010.

[35] A. Yates, M. Cafarella, M. Banko, O. Etzioni, M. Broadhead, and S. Soderland. Textrunner: open information extraction on the web. In *Proceedings of Human Language Technologies: The Annual Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*, pages 25–26. Association for Computational Linguistics, 2007.

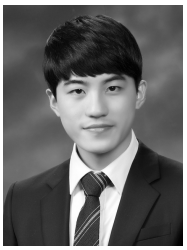
[36] X. Yin, J. Han, and S. Y. Philip. Truth discovery with multiple conflicting information providers on the web. *IEEE Transactions on Knowledge and Data Engineering*, 20(6):796–808, 2008.

[37] H. Zhang, Y. Li, F. Ma, J. Gao, and L. Su. Texttruth: an unsupervised approach to discover trustworthy information from multi-sourced text data. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2729–2737, 2018.

[38] B. Zhao and J. Han. A probabilistic model for estimating real-valued truth from conflicting sources. *Proc. of QDB*, 2012.

[39] B. Zhao, B. I. Rubinstein, J. Gemmell, and J. Han. A bayesian approach to discovering truth from conflicting sources for data integration. *Proceedings of the VLDB Endowment*, 5(6):550–561, 2012.

[40] Z. Zhu and T. Dumitras. Featuresmith: Automaticaly engineering features for malware detection by mining the security literature. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 767–778. ACM, 2016.

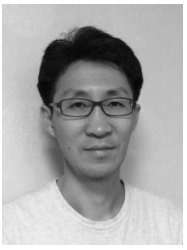


Jinwoo Kim is a Ph.D. student in the School of Electrical Engineering at KAIST. He received his M.S degree in Graduate School of Information Security from KAIST, and his B.S degree in Computer Science and Engineering from Chungnam National University. His research topic mainly focus on Software Defined Networking (SDN) security, designing a network security system, and an applied network theory.

Phillip Porras received his M.S. degree in Computer Science from the University of California, Santa Barbara, CA, USA, in 1992. He is an SRI Fellow and a Program Director of the Internet Security Group in SRI's Computer Science Laboratory, Menlo Park, CA, USA. He has participated on numerous program committees and editorial boards, and participates on multiple commercial company technical advisory boards. He continues to publish and conduct technology development on numerous topics including intrusion detection and alarm correlation, privacy, malware analytics, active and software defined networks, and wireless security.

Vinod Yegneswaran received his A.B. degree from the University of California, Berkeley, CA, USA, in 2000, and his Ph.D. degree from the University of Wisconsin, Madison, WI, USA, in 2006, both in Computer Science. He is a Senior Computer Scientist with SRI International, Menlo Park, CA, USA, pursuing advanced research in network and systems security. His current research interests include SDN security, malware analysis and anti-censorship technologies. Dr. Yegneswaran has served on several NSF panels and program committees of security and networking conferences, including the IEEE Security and Privacy Symposium.

Hyeonseong Jo is a Ph.D. student in the Department of Information Security at KAIST working with Dr. Seungwon Shin in NSS Lab. He received his B.S degree in Computer Engineering from Ajou University in Korea. He received his M.S. degree in Information Security from KAIST. His research interests include SDN security and cyber threat intelligence.



Seungwon Shin is an associate professor in the School of Electrical Engineering at KAIST. He received his Ph.D. degree in Computer Engineering from the Electrical and Computer Engineering Department, Texas A&M University, and his M.S degree and B.S degree from KAIST, both in Electrical and Computer Engineering. Before joining KAIST, he has spent nine years at industry, where he devised several mission critical networking systems. His research interests span the areas of SDN security, IoT security, and Botnet analysis/detection.