Extractor: Extracting Attack Behavior from Threat Reports

Kiavash Satvat
University of Illinois at Chicago
ksatva2@uic.edu

Rigel Gjomemo
University of Illinois at Chicago
rgjome1@uic.edu

V.N. Venkatakrishnan
University of Illinois at Chicago
venkat@uic.edu

2021 IEEE European Symposium on Security and Privacy (EuroS&P) | 978-1-6654-1491-3/21/$31.00 ©2021 IEEE | DOI: 10.1109/EUROSP51992.2021.00046

***Abstract*—The knowledge on attacks contained in Cyber Threat Intelligence (CTI) reports is very important to effectively identify and quickly respond to cyber threats. However, this knowledge is often embedded in large amounts of text, and therefore difficult to use effectively. To address this challenge, we propose a novel approach and tool called EXTRACTOR that allows precise automatic extraction of concise attack behaviors from CTI reports. EXTRACTOR makes no strong assumptions about the text and is capable of extracting attack behaviors as provenance graphs from unstructured text. We evaluate EXTRACTOR using real-world incident reports from various sources as well as reports of DARPA adversarial engagements that involve several attack campaigns on various OS platforms of Windows, Linux, and FreeBSD. Our evaluation results show that EXTRACTOR can extract concise provenance graphs from CTI reports and show that these graphs can successfully be used by cyber-analytics tools in threat-hunting.**

I. INTRODUCTION

Cyber Threat Intelligence (CTI), as commonly reported in technical reports, whitepapers, blogs, and newsgroups, is a valuable source of information about cyber-attacks. These reports describe many aspects of an attack in natural language, including the sequence of actions, effects on the system under attack, and Indicators of Compromise (IOC). The knowledge contained in CTI reports is essential for cyber operations and response personnel, system administrators, as well as vendors of intrusion detection and prevention products.

Previous studies [52], [45], [89] utilize various NLP techniques to automatically extract knowledge available in CTI reports in the form of IOCs (i.e., [52], [89]) and threat actions (i.e., [45]). While these works provide a good starting point towards automated extraction of threat elements (IOCs and threat actions) from CTI reports, they do not extract the relationships between IOCs and threat actions, in order to provide a comprehensive view of the attack behavior. Such attack behavior extraction is essential in threat-hunting activities. In particular, extracting attack behavior and the attack's big picture requires extracting the entities involved (e.g., files and sockets), actions (e.g., system calls), the causal and temporal dependencies between them, as well as information flow between the entities. Extracting the attack behavior requires an approach that is able to understand *"who did what to whom", "when" and "where"* from the natural text. This task presents several challenges.

Challenge 1: Verbosity. Threat reports are infused with a significant amount of irrelevant text; often, only a small portion of the report describes attack behavior. For instance, a description of the malware's geographical origin, though interesting, does not contribute to the description of the malware behavior in a system.

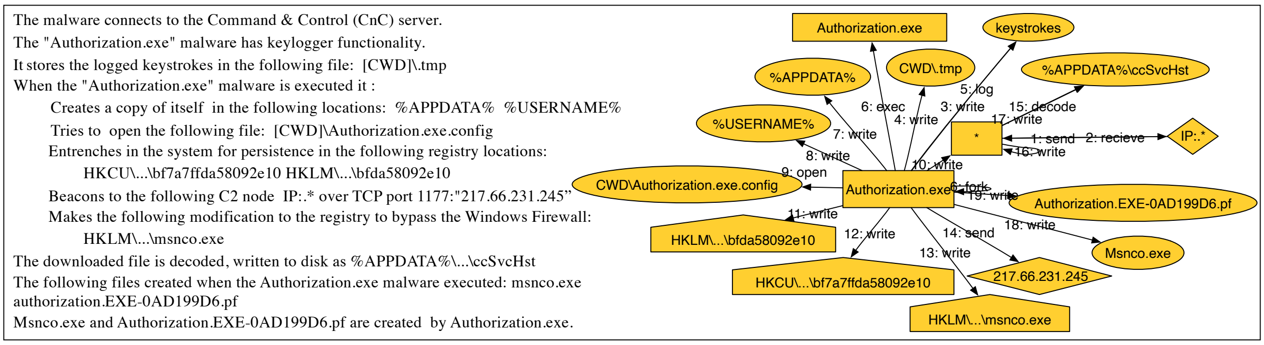
Challenge 2. CTI text complexity An important assumption of the previous approaches is that the text structure of CTI reports is (a) relatively simple [52] or (b) that it follows a specific grammatical structure [45] or (c) assuming some patterns in describing concepts [88] or (d) considering stable grammatical relations in the presentation of the sentence in the form of subject, verb and object [52], [45]. While these assumptions do not interfere with the goal of these works to extract IOCs and threat action in isolation, in fact, the majority of CTI reports contain much more complex domain-specific contexts (see Section II), which makes the extraction of attack behavior and causal inference more challenging. The CTI reports' syntactic and semantic complexities, the prevalence of technical terms, and lack of proper punctuation in these reports [62] can easily impact the interpretation of the report and extraction of attack behavior.

Challenge 3. Relationship Extraction. IOCs and threat actions can be extracted using approaches like string matching and classifiers, as suggested by [52] and [45]. However, extracting the big picture, while maintaining concise causal, temporal, and information flow of the attack throughout the report is far more complex and challenging. In fact, accurately interpreting the complex logic in technical reports is known to be an open problem in NLP [62].

In this paper, we introduce EXTRACTOR[1], which addresses these challenges. The *main goal* of EXTRACTOR is to concisely extract the full picture of the attack behavior from the technical reports in the form of a graph. EXTRACTOR overcomes the first challenge by proposing a novel text summarization approach that discerns the attack behavior text from the rest. To overcome the second challenge and to optimize overall system performance, EXTRACTOR uses a set of techniques to transform a highly complex text into a more consumable form. To address the third challenge, EXTRACTOR uses a novel approach Semantic Role Labeling (SRL), which allows us to extract the attack behavior and subject, object, and actions of the sentence by inferring the fact of *"who did what to whom", "when" and "where"* (details of these steps discussed in Sections II and III). Finally, the result of SRL in the final step is presented in the form of a *graph* describing the

[1] https://github.com/ksatvat/Extractor

© 2021, Kiavash Satvat. Under license to IEEE.
DOI 10.1109/EuroSP51992.2021.00046



The malware connects to the Command & Control (CnC) server.
The "Authorization.exe" malware has keylogger functionality.
It stores the logged keystrokes in the following file: [CWD]\.tmp
When the "Authorization.exe" malware is executed it :
    Creates a copy of itself in the following locations: %APPDATA% %USERNAME%
    Tries to open the following file: [CWD]\Authorization.exe.config
    Entrenches in the system for persistence in the following registry locations:
        HKCU\...\bf7a7ffda58092e10 HKLM\...\bfda58092e10
    Beacons to the following C2 node IP:.* over TCP port 1177:"217.66.231.245"
    Makes the following modification to the registry to bypass the Windows Firewall:
        HKLM\...\msnco.exe
The downloaded file is decoded, written to disk as %APPDATA%\...\ccSvcHst
The following files created when the Authorization.exe malware executed: msnco.exe
authorization.EXE-0AD199D6.pf
Msnco.exe and Authorization.EXE-0AD199D6.pf are created by Authorization.exe.

Fig. 1. The report (the left) is a free adaptation from the njRAT [80], where the irrelevant sentences are removed. This example demonstrates the language complexities, which will be discussed throughout the paper. In the corresponding provenance graph (the right), nodes signify system entities, and the edges point to system calls. The rectangle, oval, pentagon, and diamond represent the file, process, registry, and socket, respectively.

attack's steps, artifacts, the causal information flow between the entities involved.

In addition to the main goal of extracting the full attack picture, EXTRACTOR follows two more goals:

Goal 1: Actionable Intelligence. We want to automatically construct what we call *actionable* intelligence. We want to extract from a CTI report only information that is *ready* to be used for *detection*, or *threat hunting* without needing further actions or processing from people or tools. This means that the attack behaviors we extract from the text must be *observable* in the system audit logs and can be effectively used for threat detection. This is an important goal for every approach that extracts attack information from CTI reports. In fact, we envision the deployment of EXTRACTOR as a first step in a *threat hunting* operation.

Goal 2: Process a Large Number of CTI Reports accurately. We want to process a large number of CTI report, blogs, and attack descriptions from threat detection centers. Accomplishing this goal would enable analysts to automatically tap from a vastly larger source of knowledge than it is currently possible.

Applications of EXTRACTOR. As has been widely demonstrated, the presence of a concise attack behavior description is preferable to have a simple collection of IOCs in detecting threats [59], [84], [39], [55], [50], [56]. EXTRACTOR is able to build graph representations that represent such concise description from CTI reports, thus guiding intrusion detection and threat hunting systems. Another envisioned use of EXTRACTOR is that of extracting information from a variety of CTI sources related to the same attack in different organizations. This is to obtain a complete picture of how the same malicious actor might behave in different scenarios.

EXTRACTOR surpasses the state of the art significantly by making several important contributions. In particular, EXTRACTOR: 1) significantly expands the range of CTI reports that can be processed, 2) extracts significantly more complex details than the previous studies (e.g., [52], [45], [89]); this includes extraction of causal dependency and temporal order of attack, 3) implements a novel application of extracting semantic relationships among artifacts of an attack that enables it to obtain a much better picture of the attack, 4) implements several novel applications of text simplification and reduction (or summarization) that enable condensing the text without losing useful information.

This paper is organized as follows. In Section II, we provide a more detailed description of the problem and some background information. In Section III, we describe our approach in detail. In Section IV, we give a short overview of the implementation and different tools used. Section V presents the evaluation. Section VI provides a discussion, while Sections VII and VIII contain related work and conclusions, respectively.

II. Problem and Background

A. Problem Description

As mentioned in the introduction, the main goal of this paper is to extract *actionable* graphs representing attack behavior from generic CTI reports. By *actionable* we refer to the important goal of using the extracted knowledge as a signal in *threat hunting*. We refer to these graphs as *provenance graphs*. Provenance graphs are a common representation of kernel audit logs [49], [48]. They represent events (system calls) in a system as edges between entities (processes, files, sockets). Provenance graphs have recently been successfully used for threat detection and forensic analysis in a large number of studies [41], [59], [84], [39], [55], [50], [56].

An example of the text contained in CTI reports, inspired by the njRAT attack [80], is shown in Figure 1. This example will be used throughout the paper to illustrate different aspects of our approach. An example of the corresponding provenance graph extracted from that text is also shown in Figure 1 on the right side. As can be noticed, the provenance graph contains nodes that represent entities (processes, registry keys, etc.) involved in the attack and edges that represent the actions carried out by those nodes. In addition, the names of the nodes are such that can be observed in the audit logs, and edges connecting the nodes represent system calls that also appear in the audit logs (goal 1: actionable intelligence). In addition, the graph contains only attack behavior-related nodes and no other information (main goal of full attack picture and conciseness). We note that the natural text in Figure 1 does not have any particular structure (goal 2: process CTI reports written in natural language).

There are several challenges in extracting concise and *actionable* provenance graphs from CTI reports written in natural language. First, we need to distinguish attack behavior text from the rest of the report. This implies extracting from the natural text only the kind of relations that describe attack behavior and that can be *observed* in audit logs, while filtering out the rest of the text. Therefore, we need to understand the relations and actions occurring among system entities mentioned in the text to map those actions to system calls, which are represented as edges in the provenance graphs. Second, we need to overcome CTI text complexities, which may impact our graph extraction. This implies resolving different kinds of ambiguities and complexities present in natural language writing. We describe the challenges that must be solved in more detail next.

B. Challenges

Verbosity. In general, CTI reports can be verbose. Sentences containing useful information may be nested inside the text that is not strictly related to the attack, e.g., introductory details. For instance, out of 42 pages DustySky report [7], only 11 sentences describe the actual attack behavior that can be observed in audit logs. We separate useful content from non-useful using a novel *summarization* technique (See III-C).

CTI Text Complexity. The language used in the cybersecurity domain has several peculiarities that NLP tools/techniques (developed for more generic domains) often struggle with. This makes it challenging to use these tools as they are. We list some of these peculiarities below.

Punctuation. Many CTI reports do not use sentence-ending characters `'.,!,?'` to delimit sentences. This makes it hard for the popular NLP toolkits, such as Stanford [14], NLTK [54], and spaCy [9], to understand the *real* sentence boundaries in CTI reports, resulting in texts with long sentences, each of which contains several shorter sentences. For instance, from the observation of 4020 threat reports from the Microsoft threat report center, we notice that writers tend to pack many actions within one sentence, therefore making the average sentence length equal to 52 words (with some examples as long as 313 words per sentence). In contrast, the average English text on which NLP tools are usually trained and designed for, contains approximately 14.4 words per sentence [5].

Domain-specific words. Words denoting objects in the cybersecurity domain may have different meanings and contexts from words used in the common English language, on which NLP tools are trained. For instance, IP addresses, paths, process names, system call names, and many other terms often are misunderstood by common NLP tools. This challenge must be met by a mechanism that brings domain insight to assign meanings to the terms.

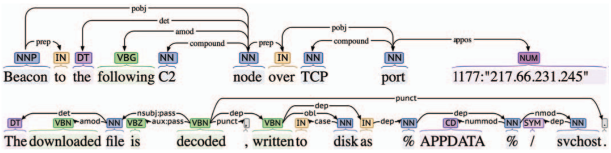

pobj
det
amod
compound
prep
prep
pobj
compound
appos
NNP IN DT VBG NN NN IN NN NN NUM
Beacon to the following C2 node over TCP port 1177:"217.66.231.245"
punct
dep
dep
obl
det
nsubj:pass
nmod
DT VBN amod NN VBZ aux:pass VBN punct , VBN IN case NN IN dep NN CD nummod NN SYM dep NN .
The downloaded file is decoded , written to disk as % APPDATA % / svchost .

Fig. 2. POS tags and DP tree obtained from [9] (on the top) and [13] (on the bottom). Tags inside the boxes represented POS and tags on the arches signify the dependencies. The figure shows examples of imprecise tagging of the technical cybersecurity sentences by statistical models.

Ellipsis. This term denotes a gap in a sentence that: 1) has a missing subject, or 2) has a missing object [24]. This structure is not common in natural English writing [64], but it is very common in CTI reports where attacks are described as sequences of actions. For instance, `Creates a copy of itself in the following locations` in Figure 1 represents an example of *ellipsis subject*.

Pronouns. Pronouns are very commonly used in English [70]. Ignoring pronouns may result in their appearance as nodes in the provenance graph in the place of the referent entities.

Other linguistic structures. Structural complexities and the use of various linguistic techniques such as anaphora, nominalization, and lists (III-B) can confuse common NLP tools. The overall effect is that many subjects, verbs, and objects are misclassified and unresolved.

Relationships Extraction. Overcoming the previous challenges can help to pinpoint the correct entities that are important in an attack description. The next step is to determine *"who did what to whom"*, *"when"* and *"where"* or in other words, we need to discover the relationships between process and system objects and their mapping to audit events. Current approaches related to this task, such as statistical dependency parsers are known for performance degradation on sentences drawn from domains different from that of natural English text [57], [58]. To resolve this issue, we need a more comprehensive approach that takes the sentence's semantics into account rather than only relying on the sentence's syntactic structure (i.e., pure use of dependency parsing, as used by [52], [45], [89]). As we will show in detail in the next section, to solve this challenge, we use *Semantic Role Labeling (SRL)*, a processing model that can detect semantic relationships among entities in a sentence.

Before continuing with the description of our approach, we provide a brief background on the NLP techniques that are used throughout the paper.

C. NLP Background

Part of Speech (POS). POS tagging assigns a syntactic role to each word in a sentence (e.g., noun, verb, etc). In some cases, however, POS model may fail to correctly tag words. In Figure 2, adjectives `following` and `downloaded` are incorrectly tagged as *verb (VBG/VBN)*.

Dependency Parsing (DP). DP assigns grammatical connections and dependencies between words in a sentence. Example of DP tags include *nsubj* for sentence subjects, *obj* for sentence

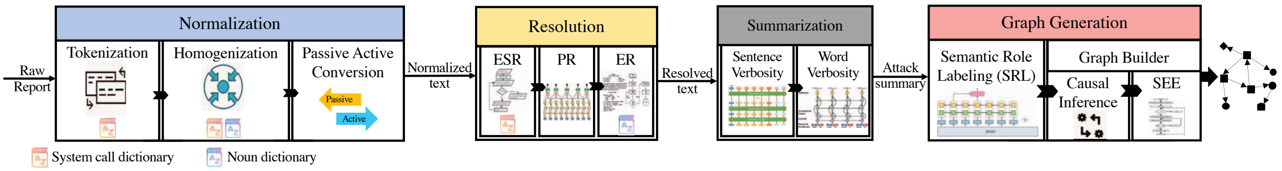


Fig. 3. An overview of EXTRACTOR architecture

TABLE I
LIST OF GENERAL ARGUMENTS USED IN SRL (BASED ON PROPBANK[67])

Label	Role (argument)	Label	Role (argument)
ARG0	Agent	ARG3	Starting point, Benefactive, Attribute
ARG1	Patient	ARG4	Ending point
ARG2	Instrument, Benefactive, Attribute	ARGM	Modifier

objects, etc. However, if the sentence complexity increases, DP may not be able to detect tags and the relations between words accurately. In addition, DP taggers may not be able to assign the correct tags, especially if they are not trained on context containing technical language. Common errors include tagging past participle forms as adjectives, verbs as nouns, etc. Another drawback of utilizing DP in our current problem is that the tags they produce only refer to grammatical relations, such as *subject, object* etc. Therefore, they cannot help in tasks that require an understanding of the semantics between different sentence components. These relationships may include temporality (when something happens), modality (how something happens), etc. In fact, a much deeper understanding is needed to accomplish our goals.

Figure 2 demonstrates examples of POS tags and DP trees driven by spaCy [9] (top) and Stanford [13] (bottom), where incorrect tags such as *following* and *downloaded* (as discussed earlier) caused the incorrect generation of DP relations as subject and objects. Another example is the verb *Beacon* at the top sentence which incorrectly tagged as a proper noun (i.e., NNP). More about POS and DP can be found at [27] and [22].

Semantic Role Labeling (SRL). SRL essentially determines *"who did what to whom"*, *"when"* and *"where"* [67]. SRL is a more recent NLP technique, which can assign semantic labels to phrases and words in a sentence, where each label specifies the *semantic role* that each phrase or word plays in the sentence in association with the predicate or verb of the sentence. In SRL, the tags assigned to sentence components are called *arguments* (denoted by ARG). Some argument examples and the corresponding semantic roles are shown in Table I.

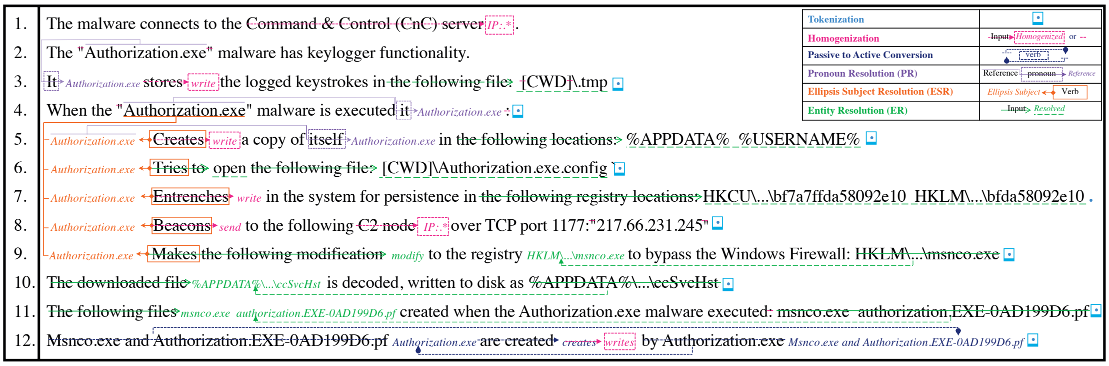
III. APPROACH

In a nutshell, EXTRACTOR operates by performing different rounds of transformations on the text to bring it from a highly complex and potentially ambiguous form to a simpler form. This simplified text is further processed to obtain a provenance graph that can be successfully used for threat detection. An overview of EXTRACTOR is shown in Figure 3. EXTRACTOR has four major components: 1) Normalization, 2) Resolution, 3) Summarization, and 4) Graph Generation. *Normalization* is responsible for an initial round of sentence simplification and transformation to a canonical form. *Resolution* resolves ambiguities in those sentences (these two components help to address CTI text complexity challenge). *Summarization* removes the portion of text that is not strictly related to the attack behavior, and that cannot be observed in the logs. Finally, *Graph Generation* is responsible for resolving the temporal and causal order among the events in the text and for building the final provenance graph (this component addresses the Relationships Extraction challenge). Some of these components may be assisted by a set of dictionaries that contain terms related to CTI language (relying on domain-specific dictionaries of concepts is a common approach in many knowledge-based NLP systems [76], [71], [78]). In particular, EXTRACTOR uses two dictionaries. First, our system call synonym dictionary, which contains verbs representing system calls (e.g., write, fork) and their corresponding synonyms. These synonyms represent the possible verbs that can be used in CTI reports and very likely refer to a system call. Second, our CTI nouns dictionary contains noun phrases commonly used in CTI reports, as well as different textual representations of the same concept. The former contains 87 verbs representing system calls, while the latter holds over 1112 common noun phrases in the CTI report. Both dictionaries are depicted in Figure 3, and will be further discussed in Section IV.

A. Normalization

To address the CTI text complexity challenge and maximize the accuracy of the techniques used by EXTRACTOR, we must first have some canonical sentence form. We achieve this through Normalization, which is responsible for breaking long and complex sentences into shorter sentences appearing in a canonical form, which is easier to process. Intuitively, we would like each sentence to express a single action so that the subject and object of the action and the action itself be easier to identify. Normalization is comprised of Tokenization, Homogenization, and Conversion. These steps perform the detection of sentence boundaries, word homogenization, and passive-to-active verb conversion, respectively. We describe each of these steps next.

Tokenization. Correctly defining sentence boundaries is very important as several techniques used by EXTRACTOR operate at the sentence level. However, existing sentence tokenizers (e.g., NLTK [54]) usually take only classic punctuation (`'.,!,?'`) into account when discovering sentence boundaries and perform poorly on CTI reports. In fact, in this domain, there is a high prevalence of long sentences containing



1. The malware connects to the ~~Command & Control (CnC) server~~ *IP:.**.
2. The "Authorization.exe" malware has keylogger functionality.
3. ~~It~~ *Authorization.exe* ~~stores~~ *write* the logged keystrokes in ~~the following file:~~ [CWD]\.tmp
4. When the "Authorization.exe" malware is executed ~~it~~ *Authorization.exe* ~~:~~
5. *Authorization.exe* ~~Creates~~ *write* a copy of ~~itself~~ *Authorization.exe* in ~~the following locations:~~ %APPDATA% %USERNAME%
6. *Authorization.exe* ~~Tries to~~ open ~~the following file:~~ [CWD]\Authorization.exe.config
7. *Authorization.exe* ~~Entrenches~~ *write* in the system for persistence in ~~the following registry locations:~~ HKCU\...\bf7a7ffda58092e10 HKLM\...\bfda58092e10 .
8. *Authorization.exe* ~~Beacons~~ *send* to the following ~~C2 node~~ *IP:.** over TCP port 1177:"217.66.231.245"
9. *Authorization.exe* ~~Makes the following modification~~ *modify* to the registry *HKLM\...\msnco.exe* to bypass the Windows Firewall: ~~HKLM\...\msnco.exe~~
10. ~~The downloaded file~~ *%APPDATA%\...\ccSvcHst* is decoded, written to disk as ~~%APPDATA%\...\ccSvcHst~~
11. ~~The following files~~ *msnco.exe authorization.EXE-0AD199D6.pf* created when the Authorization.exe malware executed~~:~~ ~~msnco.exe authorization.EXE-0AD199D6.pf~~
12. ~~Msnco.exe and Authorization.EXE-0AD199D6.pf~~ *Authorization.exe* ~~are created~~ *creates* *writes* ~~by Authorization.exe~~ *Msnco.exe and Authorization.EXE-0AD199D6.pf*

Fig. 4. Transformation steps to turn a CTI report into a digestible form. A free adaptation from the njRAT [80] where irrelevant sentences are removed. Lines with pointer signifies the reference and the strike-lined pointer shows the original phrase and its substituted output. The figure best appears in color.

multiple actions and non-standard sentence delimiters. For instance, in Microsoft threat reports, the average sentence length is almost four times higher than that of common English sentences.

To solve this problem, we design an enhanced tokenizer specialized for CTI reports. In particular, in addition to the classic sentence delimiters, our tokenizer uses new lines, bullet points, enumeration numbers, and titles and headers, as *sentence delimiters* to partition long sequences into sets of shorter ones. After breaking long sentences into shorter sequences of words, each short sequence is 'promoted' to a sentence if it satisfies one of the following cases; 1) the sequence starts with a capitalized subject, it contains all the components necessary to form a complete sentence (subject, predicate, object), and the preceding and subsequent sequences also form complete sentences; 2) the sentence starts with a verb contained in the system calls dictionary, it contains all the components necessary to form a complete sentence minus the subject, and the preceding and subsequent sequences also form complete sentences. The latter case represents the common phenomenon (in CTI reports) of Ellipsis Subject (see Section II-B). If none of the above two cases is satisfied, we consider the sentence as an unbreakable full sentence.

As an example of this procedure, consider Figure 4, which illustrates several techniques described in the paper. In this figure, the long sentence spanning lines 4-9 in Figure 4 is first partitioned into shorter sequences (one per line in the figure). Next, each sequence is tagged by a POS tagger and DP, and checked if it satisfies one of the two conditions above. In the figure, the sequence at line 4 satisfies the conditions of the first case, while the sequences at lines 5-9 satisfy the conditions of the second case (ellipsis subject).

The result of the tokenizer is a set of shorter sentences that is more likely to describe a single action.

Homogenization. CTI reports often contain constructs and synonyms that can introduce ambiguities and impact the final results' quality. For example, `C2`, `C&C`, and `Command and Control` are different representations of the same entity, while verbs like `stores`, `saves` may represent an action that corresponds to a `write` system call. *Homogenization* is the process by which multiple textual representations of the same concept are replaced by the same textual representation.

We perform *Homogenization* for noun phrases and verbs using two specially built dictionaries, which map different jargon and synonyms of nouns and verbs present in CTI reports to entities and actions that can be observed in audit logs. For instance, each among `C2`, `C&C`, `Command and Control` is mapped to `IP:.*`, which is a wildcard representing IP addresses. In the same manner, we translate verbs that are synonyms with a system call inside the system call dictionary with that system call verbs.

Homogenization significantly reduces text's heterogeneity and supports our goal of providing actionable intelligence. We decide that the single word that is chosen to represent all the other words of a synset is one that is highly likely to be as a system entity that is observable in the logs or a *system call*.

Conversion. As the last step of text normalization, EXTRACTOR converts passive voices to active. This conversion helps with discovering system subjects (processes) and system objects, as well as making causality inference more accurate, as discussed in Subsection III-D.

To perform this conversion, we first detect passive sentences using POS and DP tagging. This kind of sentence is predominantly represented by specific and known patterns in DP trees. For instance, consider the sentence `the downloaded file is deleted by the malware`. In the DP tree, *is* is tagged as an auxiliary (and passive) verb, `deleted` as a verb and head of the DP tree, `the downloaded file` is a noun phrase that is the subject of a passive voice (*nsubjpass*) and `by malware` is the object *(obj)*. Note that in some cases, the agent does not appear in a passive sentence but is implied. For instance, in line 10 in Figure 4, the agent is the malware, but no references to it appear in the sentence. Using these patterns, EXTRACTOR can detect passive sentences and distinguish between passive sentences with explicit agents and those with implicit agents. In the former case, it switches the agent and the subject, and it conjugates the passive verb to an active verb.

The final result of this step is that long sentences are transformed into short ones in an active form, likely to express one action per sentence.

B. Resolution

After *Normalization*, *Resolution* reconciles implicit references that refer to the same entity into the actual referent. These implicit references must be made explicit for two reasons. First, implicit references reduce the accuracy of the subsequent steps and make the final provenance graph ambiguous and imprecise. Second, audit logs contain only explicitly named entities, and every threat hunting approach cannot match system processes to pronouns and other implicit references. More thorough and fascinating discussions on such linguistic structures can be found at [81], [65], [83].

Ellipsis Subject Resolution (ESR). As discussed in Section II, ellipsis subject is a linguistic structure where a sentence's subject is not present. This kind of structure is shared in a large number of CTI reports and used for describing a sequence of actions carried out by the same actor (process or attacker)- Section V presents the popularity of this phenomenon in various sources. The omitted subjects confuse the state-of-the-art NLP toolkits, thus resulting in the loss of the narrative sequence and the story relationships (subject and object of an action). All the actions described in lines 5-9 in Figure 4 are examples of ellipsis subjects.

To address this problem, we developed an Ellipsis Subject Resolver (ESR) module. This module utilizes POS and DP parsing along with the system calls dictionary. The first step in resolving this problem is the detection of sentences with missing subjects. This step uses POS and DP together with the system calls dictionary, as was described in the discussion about the Tokenizer (Subsection III-A). Once this kind of sentence is detected, ESR builds a list of candidate subjects among the entities appearing in the sentences preceding the current sentence. Next, the module picks the most probable candidates from the list based on the distance (computed as the number of sentences) of that candidate from the sentence with the missing subject. In particular, the closer candidate has a higher probability of being picked. For instance, in Figure 4, the subject is missing in the sentences in lines 5-9. The ESR module detects the subjects and other objects in the previous sentences, and it chooses the pronoun *it* occurring right before the colon as the subject.

Pronoun Resolution (PR). Pronoun resolution is the process by which pronouns are mapped and substituted to the antecedent entities that they refer to. Processing documents (building a provenance graph) without PR can result in the appearance of several nodes (i.e., pronouns) for a single entity. For instance, in Figure 4, the pronoun `it` and `itself` in lines 3 and 5 should be replaced with the actual subject `Authorization.exe`.

To resolve pronouns, we adapt a popular coreference resolution model, NeuralCoref [10]. We noticed that this model works best in resolving pronouns in the CTI reports domain, especially after the previous steps of ESR, and Tokenization. Figure 4, lines 4, 5, and 6 demonstrate the resolved pronouns (i.e., it and itself) and their corresponding reference (`Autorization.exe`).

Entity Resolution (ER). Entity resolution is the process by which noun or verb phrases that refer to another entity inside the same sentence are substituted by that entity or are eliminated as redundant. This is a vast task to perform in general, however, we point out that we are interested only in extracting *actionable* information and, therefore, can focus on performing ER only on entities and actions that are likely to appear in audit logs. In fact, from a preliminary observation of a large number of CTI reports, we noticed that redundancies among those entities and actions appear under mainly three distinct linguistic forms:

Anaphora. An anaphora is the use of a word or pronoun to refer back to another word or phrase that was previously used in the sentence to avoid repetition. For instance, in line 11 of Figure 4, `The following files` refers to `mscno.exe authorization.EXE-0AD199D6.pf`. This form is prevalent in CTI reports, where it is used to describe lists of entities participating in some common action.

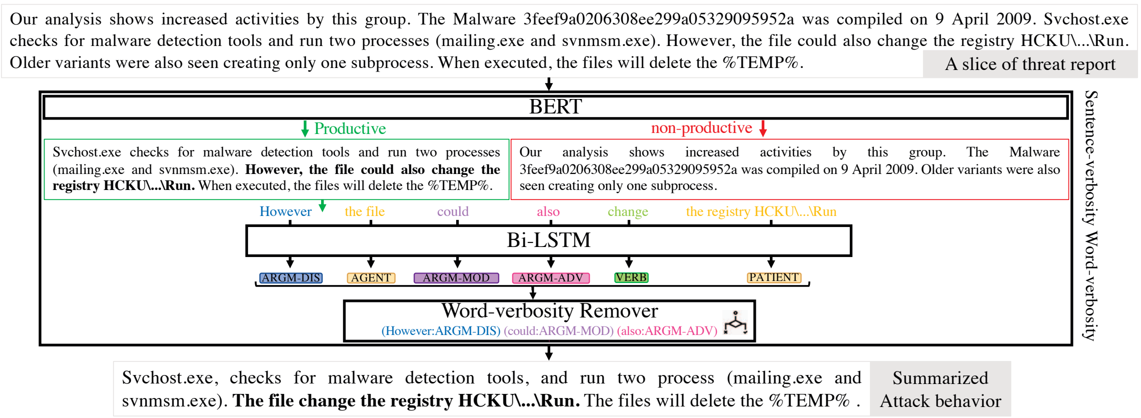
Nominalization. This is a form where an auxiliary verb is used together with a noun in place of a verb. For instance, *makes a modification* in place of *modifies*. This form is often used with actions that represent system calls. In particular, it appears approximately 3524 times in the TrendMicro and 1261 times in Microsoft blogs. Another similar form related to system calls appears as an auxiliary verb followed by an actual verb related to a system call, e.g., *tries to open* instead of *opens* in Figure 4.

To resolve these cases, we use a combination of POS tagging and DP with domain knowledge contained in *CTI nouns* dictionary or in a corpus of common phrases appearing in each case (e.g., *the following files* is a common anaphora). In particular, if one of the above three forms is detected in the text, we retrieve the DP and POS tags of the other words in the vicinity of that form and check that they follow specific patterns. In particular, for anaphoras, we check that a list of noun phrases follows the main sentence where the anaphora appears and replace the anaphora with the noun phrases. For nominalizations, we check that the noun present in the corpus is the object of a preceding auxiliary verb and replace that noun with its verb form (e.g., `makes the modification` → `modifies`). For auxiliary verbs, we detect if an infinitive form precedes a verb that may represent a system call and replace the whole phrase with the actual verb (`tries to open` → `open`).

After the *Resolution* step is completed, the text consists of sentences having explicit subjects, objects, and verbs. The amount of text is also reduced somewhat by the ER module. However, the major text reduction step is executed after *Resolution* and is described next.

C. Text Summarization

To reduce verbosity and obtain a concise description of the attack behavior that can be directly used to detect the attack, a significant amount of superfluous text must be removed.



Our analysis shows increased activities by this group. The Malware 3feef9a0206308ee299a05329095952a was compiled on 9 April 2009. Svchost.exe checks for malware detection tools and run two processes (mailing.exe and svnmsm.exe). However, the file could also change the registry HCKU\...\Run. Older variants were also seen creating only one subprocess. When executed, the files will delete the %TEMP%.

Svchost.exe checks for malware detection tools and run two processes (mailing.exe and svnmsm.exe). **However, the file could also change the registry HCKU\...\Run.** When executed, the files will delete the %TEMP%.

Our analysis shows increased activities by this group. The Malware 3feef9a0206308ee299a05329095952a was compiled on 9 April 2009. Older variants were also seen creating only one subprocess.

Svchost.exe, checks for malware detection tools, and run two process (mailing.exe and svnmsm.exe). **The file change the registry HCKU\...\Run.** The files will delete the %TEMP% .

Fig. 5. The architecture of the Text summarizer. The module reduces the verbosity using BERT, BiLSTM, word-verbosity remover.

Ideally, only the sentences that describe actions that may be observed in the audit logs should be preserved. To do this, EXTRACTOR must understand which sentences strictly describe attack behavior and which sentences do not. Previous related work uses *topic classification* [52], [45] to identify *topic*-related context among out-of-domain contexts (e.g., advertisement text versus technical text). While these approaches are successful in separating irrelevant content (such as ads) from technical content, they are not powerful enough to separate the latter into *behavioral* content that describes observable attack actions from other "technical" content, which serves as an introduction or context description. We refer to this problem as *sentence verbosity*. An example of *sentence verbosity* is shown in Figure 5. In the figure, the text of the report is shown at the top. The sentences in the box on the top left corner, labeled by *Productive*, contain a description of the malware's actual behavior, which can be observed in audit logs, and which can, therefore, be useful for detection. The sentences on the top right corner, labeled by *non-productive* contain the complementary description of the malware but no actions that can be observed in audit logs. Even though the two text portions are technical in nature and about the same *topic*, we are only interested in the *productive* text and want to remove the *non-productive* one.

Another problem that needs to be solved is what we refer to as *word verbosity*. In particular, inside each sentence, there usually appear word constructs, such as adverbial and adjectival phrases, which do not contribute to the behavior description and can be safely removed (e.g., `However`, `could`, and `also` in the figure).

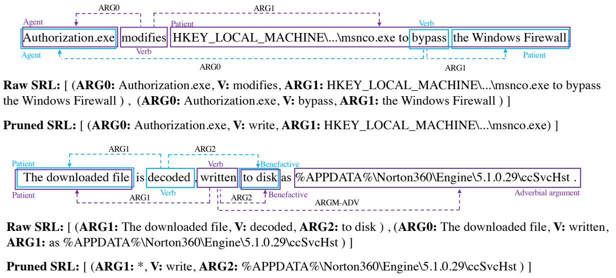
To deal with these problems, we design a two-step approach. This approach is shown in Figure 5 and is composed of a BERT classifier, which deals with *sentence verbosity*, and a BiLSTM network, which deals with *word verbosity*.

Sentence Verbosity. To distinguish sentences that describe actual threat behavior from the ones that do not represent threat behavior, we need to go beyond *topic classification* and have a deeper understanding of the text. Intuitively, *productive* sentences express more "direct" connections between the subject and the object than the other sentences. Thus, to classify these connections, a linguistic model of the text must build a finer-grained representation of the words' context.

Currently, one of the best models to build such fine-grained representation is BERT (Bidirectional Encoder Representations from Transformer) [28]. Unlike Word2Vec [1] and GloVe [2] word representations, BERT builds contextual representations of the words that take into account both the text before a word and the text after a word. In other words, BERT considers the context surrounding each word. In addition, BERT learns embeddings for subwords, that is sub-components of a word derived from stemming. This allows the model to more effectively deal with out-of-vocabulary words. In general, this capability is beneficial for the technical cybersecurity documents, which may contain lexically complex phrases that do not appear in BERT's training set. As a result, BERT can classify sentences into productive and non-productive much better than other approaches. In particular, we labeled 8,000 threat sentences under two classes of productive and non-productive, and trained BERT on this set. The results are promising and are shown in detail in Section V.

Word Verbosity. The second step of the Text Summarizer removes unnecessary words from the *productive* sentences that it receives as input from BERT. It is composed of two phases, a BiLSTM network that derives the semantic roles of the sentence components and a word remover phase. We found that BiLSTM works best for this purpose since it can handle long-distance dependencies that appear in technical documents.

After a sentence is processed by a BiLSTM network, its components are tagged as *Agent*, *Patient*, and *Action*, and other types of arguments (e.g., in Figure 5 the word *However* is labeled as ARGM-DIS, a discourse marker that connects a sentence to a preceding sentence). In the next phase, the unnecessary sentence components are removed. In theory, this can be done only by keeping the *Agent*, *Action*, and *Patient* components of the sentence. However, in certain cases, this



Raw SRL: [(**ARG0:** Authorization.exe, **V:** modifies, **ARG1:** HKEY_LOCAL_MACHINE\...\msnco.exe to bypass the Windows Firewall) , (**ARG0:** Authorization.exe, **V:** bypass, **ARG1:** the Windows Firewall)]

Pruned SRL: [(**ARG0:** Authorization.exe, **V:** write, **ARG1:** HKEY_LOCAL_MACHINE\...\msnco.exe)]

Raw SRL: [(**ARG1:** The downloaded file, **V:** decoded, **ARG2:** to disk) , (**ARG0:** The downloaded file, **V:** written, **ARG1:** as %APPDATA%\Norton360\Engine\5.1.0.29\ccSvcHst)]

Pruned SRL: [(**ARG1:** *, **V:** write, **ARG2:** %APPDATA%\Norton360\Engine\5.1.0.29\ccSvcHst)]

Fig. 6. Examples of semantic roles and relations. Roles are generated according to the PropBank annotations. Words on the arch present the labels and roles are signified by words on the top/bottom of the rectangles.

approach would remove important information. For instance, a sentence such as *when malware.exe is executed* may be labeled as ARGM-TMP (a temporal marker), and removed by a naive approach, leading to the removal of an important part of the attack. To make this second phase more precise and not remove sentence components that may contain important objects like *malware.exe*, we use the *System Entity Extractor (SEE)* which will be introduced in Section III-D2. In particular, a sentence component that is tagged for removal will be removed if it does not contain any entities that can be generated by the rules of the *SEE* component.

Text Summarization is one of the central components of EXTRACTOR. It is responsible for greatly reducing the text's complexity and quantity while keeping the most important sentences that describe observable behavior.

D. Graph Extraction

After the previous steps, the resulting text is in a form where the system subjects (e.g., process), objects (e.g., file, socket), and actions (e.g., exec) are explicit, well ordered, and a large part of the superfluous text is eliminated. In this last step, EXTRACTOR addresses the challenge of relationship extraction to extract a provenance graph from the simplified text.

Even though the text at this step is very simple, a naive graph extraction that assigns nodes to subjects and objects, and edges to the verbs would create ambiguous and large graphs. This is because several roles and relationships between subjects and objects may be expressed in the same sentence. To deal with this problem, we use Semantic Role Labeling (SRL) and a set of rules to extract the causality relations and directions of information flow. These are described next.

1) Semantic Role Labeling (SRL): As mentioned in Section II, SRL is a technique that discovers the semantic roles in the sentence. To give an intuitive overview of the power of SRL, consider the two examples in Figure 6, one in the active and one in the passive form. SRL is able to extract two roles from each sentence (denoted by *Raw SRL*) and understand which noun is the patient (that is the one the action falls on, denoted by ARG1) and which is the agent (the noun carrying the action, denoted by ARG0). For the purpose of our discussions, a *SRL role* can be thought of as an action. SRL is, therefore, able to correctly associate each component in a sentence with a semantic tag.

EXTRACTOR considers all the possible arguments related to a verb detected by SRL as potential subjects or objects of the attack and then prunes out those which are not system entities. For the pruning process, we use the *SEE* module (see Section III-D2), which detects possible system entity names (e.g., file or process names, IP, and registry keys). In particular, the *SEE* module analyzes each node and prunes out the whole node or part of the node name that does not match one of the regular expressions or application names. The result of the pruning in Figure 6 is denoted by *Pruned SRL*.

Actions to System Calls Transformation. After SRL, EXTRACTOR performs a second *homogenization* step over the verb roles detected by the SRL module. This step is necessary to correct eventual errors due to POS and DP taggers' inefficiency, which might allow verbs to slip through and remain in their original form (untranslated to system calls). After this step, we prune away those roles created by SRL that do not represent a system call action. For instance, after this second pruning step, the second role related to the verb `bypass` in the top half of Figure 6 is pruned out.

2) Graph Builder (GB): The final step of our approach is to construct the graph from the output of SRL. The *GB* operates in two steps. First, it *merges* the *SRL arguments* that have the same text into the same node, and using *SEE* prunes out words that are not system entities. Next, *GB* builds the graph using:

1) *Node-edge-node triples.* For every sentence, if it has at least three roles including a verb role (a system call representation as a connector) and two nodes, GB generates the edge and node pairs.
2) *Edge direction.* EXTRACTOR determines the direction of the edges by using a small map of edge directions associated with the system calls dictionary. This is discussed in more detail the *Causal Inference* paragraph later in this section.

System Entity Extractor (SEE). We developed the *SEE* module to extract concise nodes that represent system entities from the roles generated by SRL, and to prune out the futile part of speeches that cannot constitute possible system entities. SEE detects possible system entity names (e.g., file or process names, IP, registry keys) using over 32 different regular expressions and a database of application names or well-known processes. In particular, the *SEE* module analyzes each noun phrase and prunes out the whole phrase or the part of it that does not match one of the regular expressions or application names. For example, in the sentence, `The malware deleted the regex.exe.`, *SEE* prunes out the (ARG0) into asterisk (* - which in query processing systems will be inferred as any) and turns (ARG1) into regex.exe. This step is essential to have concise and accurate names for the system entities that can be used to search the audit logs for threat detection. This module also defines the shape of the nodes in the final graph, where the rectangle, oval, pentagon, and diamond represent the file, process, registry, and socket, respectively.

Causal Inference. This step determines the correct direction of the edges in the graph to represent causality and information flow among nodes. To infer this direction, it uses a mapping of system calls to the direction of system flows. The mapping contains entries that associate with each system call the direction of the edge between the subject and the object (e.g., for the *send* system call the flow goes from subject to object, while for the *recv* it goes from object to subject). Besides, this step addresses negated verbs, which may appear in CTI reports. In fact, in the casual inference, we detect the negation using SRL tags and purge the negated roles if there are no conditional clauses that influence the role. For instance, `svchost.exe does not create explorer.exe` will be purged as no conditional clauses influences this sentence.

The output of the last step is a provenance graph that clearly shows the entities that participate in and are affected by the attack as nodes, as well as the system calls connecting them as edges. An example of such a graph related to the running example is shown in Figure 1.

IV. Implementation

In this section, we briefly describe some additional implementation details, tools, and techniques used by EXTRACTOR.

NLP toolkits. We used a combination of various state of the art NLP toolkits to implement our approach. These include the *spaCy* POS and DP tagger, *NLTK*, and *Stanford* [9], [13], [54]. We use *SpaCy* in *Tokenization, Homogenization, Resolution*, and *Passive to Active Conversion* steps to determine the POS and DP tags of the different sentence components. In particular, we used the large pre-trained statistical model version 2 [82] of spaCy as the model outperforms the other statistical models in dealing with CTI reports.

Tokenization. Our sentence tokenizer is built on top of the NLTK sentence tokenizer. NLTK (Natural Language ToolKit) is a common NLP toolkit, containing several libraries and statistical natural language processing developed for the English language. We chose NLTK because we found that it works better than others (spaCy, Stanford) and it is more consistent in dealing with text in the CTI domain.

Text Summarization. We used a 12 hidden layer BERT [28] to discern the *productive* sentences from the *non-productive* ones. To train our model, we used 8,000 labeled sentences. To understand the words' roles in the text summarizer, we used a re-implementation of a deep BiLSTM model [40]. Since the model was not fine-tuned to handle cybersecurity sentences, we trained the model using 3,000 manually labeled sentences.

SRL. To implement SRL, we use the method described on [79], deployed by [34], which is becoming increasingly popular in the NLP area. To adopt the system and receive more precise output in the cybersecurity domain, we further retrain the model with 2,000 cybersecurity sentences related to the areas in which we notice that the system fails to predict the roles properly. For further completeness, we evaluated and presented the result of our retraining (see Section V-D).

TABLE II
SYSTEM CALL VERBS AND THEIR CORRESPONDING SYNONYMS IN SYSTEM CALL DICTIONARY.

System call	Synonyms
Write	write, form , entrench, place, exfiltrate, deploy, implant, drop, install, putfile, compose, create, copy, save, add, modify, append, create
Read	survey, read, gather, download, navigate, locate, get, acquire, check, detect, record, exfiltrate , extract, obtain
Unlink	unlink, delete, clear, remove, erase, wipe, purge, expunge
Send	send, transfer, post, postsinformation, move, transmit, deliver, push, redirect
Receive	receive, accept, take, get, collect
Connect	connect, click, browse, portscan, communicate
Fork	fork , clone, spawn, issue, set
EXEC	use, execute, executed, run, launch, call, perform, list, invoke, inject, open, target, resume
Exit	exit, terminate, stop, end, finish, break off, abort, conclude
MMAP	allocate, assign

Dataset and Dictionary Construction. To build dictionaries and our datasets, we used our pool of CTI reports scraped from various sources. We used different sources of namely APT report repository [6], Microsoft Threat Center [8], Symantec Security Center[16], Threat Encyclopedia [18], and Virus Radar [20] to ensure the diversity and coverage.

For our text summarizer, we annotate a balanced dataset of 8,000 sentences sampled from various sources and annotated with two categories: productive and non-productive sentences. In total, 3,800 sentences are annotated as productive, 4,200 messages are annotated as non-productive sentences. We split our dataset into 4,800 sentences for training, 1,600 sentences for validation, and 1,600 sentences for the test. We used distinct sets for test and evaluation.

We perform annotation in an iterative fashion, and three subject matter experts were involved in the annotation of our datasets. We request each annotator to annotate the collected data into one of the two categories of productive and non-productive. Then, on several discussions with the annotators, we discuss and clarify the notion of the attack behavior (i.e., productive) versus the rest (i.e., non-productive) to ensure the understanding of attack behavior is accurate. Following prior guidelines and studies (i.e., [33] and [30]), the annotation task begins in an iterative fashion. In each round, 200 messages are assigned, and disagreements are discussed with each annotator. After each round of discussions, 100% inter-annotator agreement (IAA) is achieved as measured by Cohens kappa coefficient. After three initial rounds of annotations, the annotators are assigned the remaining 7,400 sentences, where an IAA of 91% is obtained. The final round of disagreements are discussed, and labels are finalized by one of the authors of this paper.

An alternative solution to translate the verb phrases into the corresponding system call is to use tools like WordNet [60] and Thesaurus [17] (researchers like [45] have previously used this kind of approach). However, we noticed that we could achieve better results by creating a simple though effective dictionary. To build our dictionaries, similar to the process of annotating our dataset, we worked with a team of three security experts in an iterative fashion. The members were involved in reviewing and annotating 3000 randomly selected technical threat reports from various sources over a period of one year. Then, in an iterative fashion, the extracted phrases and their corresponding synonyms have been discussed and agreed. Similarly, the system calls dictionary is derived from WordNet [60] and Thesaurus [17]. These synonyms have been extracted and discussed in several discussions to assure the quality. Tables II presents this dictionary. Also, Table XIII, in Appendix, represents examples of the noun dictionary.

V. Evaluation

To evaluate Extractor, we designed three experiments, each performed on CTI reports with distinct writing styles. In the first experiment (§V-A), Extractor generates graphs from a set of public CTI reports describing real-world incidents. In the second experiment (§V-B), Extractor builds graphs from the descriptions of attacks in the DARPA Transparent Computing program [19] evaluations. Finally, in a large scale experiment (§V-C), Extractor processed 4,100 unstructured CTI reports from Microsoft Security Intelligence [8] and 11,600 reports from TrendMicro [18] to extract provenance graphs as further discussed in Section V-C.

In the first two experiments, we evaluate Extractor in two distinct ways: (1) We measure Extractor's capability in capturing all relevant attack behaviors using the ground truth present in the reports. To this end, we report precision, recall, and F1-score. (2) To demonstrate the usefulness of Extractor in supporting threat hunting, we use a threat hunting system, POIROT [59], with the graphs generated by Extractor. Finally, to evaluate the scalability of our approach, we perform a large scale experiment, which is discussed in Section V-C.

Threat Hunting. To evaluate the usefulness of the graphs generated by Extractor for threat detection, we used POIROT system [59]. This system takes as input a small provenance graph, called *query graph*, representing attack events, and searches for embeddings of that graph in a much larger provenance graph built from the audit logs of the systems under attack. The query graphs in POIROT are manually built by experts after reading CTI reports and represent the attack activities described in those reports. In our evaluation, we use the same CTI reports to automatically build graphs with EXTRACTOR and use those graphs as query graphs for POIROT. In this way, we compare graphs built by human experts and graphs built by EXTRACTOR and the usefulness of both kinds of graphs to detect threats. We define an operation of $P(G_1, G_2) = S$, where G_1 represents the graph built by EXTRACTOR, and G_2 represents the larger provenance graph representing the audit logs of the systems under attack. Next, we use POIROT to search for G_1 within G_2 and retrieve the similarity score S. If S is bigger than the POIROT threshold ($t \approx 0.3$), then G_1 is successfully located in G_2, indicating a successful detection of a threat. Otherwise, no attack has been detected. For more details on POIROT, refer to [59].

In all the experiments, we measure Extractor's false positive and false negative edges. By false positive edges, we refer to the edges included in the extracted graph, which do not represent attack activities. By false negative edges, we refer to edges that should have been included in the extracted graph.

TABLE III
Characteristics (nodes |V| and edges |E|) of the Extractor vs. manual graph and results of threat detection in CTI reports, score $P(G_1, G_2)$ and detection outcome DO.

Scenario	Manual		EXTRACTOR		Score	Do
	|V|	|E|	|V|	|E|		
njRAT [80] - fig. 7	14	14	32	32	0.4	✓
Carbanak [47] - fig. 10-(a)	10	10	22	31	0.4	✓
HawkEye [87] - fig. 10-(b)	17	34	29	31	0.4	✓
DeputyDog [61] - fig. 10-(c)	5	4	11	12	0.4	✓
DustySky [7] - fig. 10-(d)	9	10	12	21	0.6	✓
Uroburos [35] - fig. 10-(e)	12	15	19	23	0.5	✓

We point out that these notions of false positive and false negative edges refer only to the presence (or lack thereof) of nodes and edges in the final graph and not to the actual detection of the threat using that final graph. In fact, many detection tools might be able to use a small set of nodes and edges as IOCs. As a specific example, the tool we used in this paper, POIROT, employs approximate graph matching using graphs with extraneous or missing edges [59], and is robust to a certain degree of false-positive and false-negative edges.

A. Evaluation on Public CTI reports.

In the first set of experiments, we evaluate Extractor using public CTI reports. For comparison purposes, we choose the same reports chosen by the authors of POIROT [59]. This experiment allows us to 1) compare the graphs generated by Extractor and the graphs generated manually by the authors of POIROT, and 2) use POIROT to perform threat hunting using the graphs generated automatically by Extractor and see if the attack is successfully detected. In this experiment, the audit logs contain events generated by benign activities and events generated by executing the malware instances and the same attack activities described in the CTI reports in a controlled and isolated environment, as described by the authors of POIROT [59]. Table XIV, in the Appendix, provides additional details about each malware sample.

Table III represents the characteristics of Extractor graphs versus manual graphs and the result of threat detection in these public CTI reports. The first column shows the malware name, the reference to the CTI report, and the reference to the extracted provenance graph figure. The next four columns show the number of nodes ($V(G)$) and edges ($E(G)$) of the graphs manually drawn by the POIROT authors and the ones automatically generated by Extractor. As can be seen, the numbers of nodes and edges extracted by Extractor are comparable with the ones built manually. The main reasons for the difference in the number of nodes and edges are due to 1) the use of wildcards in the manual graphs (e.g., the use of *C=*.tmp* in manual instead of *[CWD]\\tmp* and *C:\Extracted\\tmp*), 2) nodes and edges that are picked by Extractor but are not presented in the manual graph (e.g., *2: exec - 10: exec*), as the human has abstracted these details away. Finally, columns six and seven present the results of threat hunting, which was obtained by conducting these

TABLE IV
PRECISION, RECALL, AND F1-SCORE OF GRAPHS GENERATED FROM THE CTI REPORTS, CALCULATED BY COMPARING EDGES IN THE AUTOMATED GRAPH AGAINST THOSE IN THE CTI REPORT GROUND TRUTHS.

Scenario	Precision	Recall	F1-Score
njRAT	0.90	1	0.95
Carbanak	0.87	1	0.93
Uroburos	0.85	0.96	0.90
DustySky	0.85	0.94	0.90
HawkEye	0.93	0.93	0.93
DeputyDog	1	0.92	0.96

malware attacks in the presence of suitable benign activities, and collecting the audit records. An approximate matching algorithm [59] was used to match the EXTRACTOR-generated graph inside a larger provenance graph generated from the audit logs of the systems under attack. In all scenarios, our detection score surpassed the detection threshold ($t \approx 0.3$), and the attack was detected successfully. In summary, through this experiment, we can conclude that the EXTRACTOR-generated graphs are as useful as human-generated graphs in threat detection.

While the results of the threat detection (Table III) using EXTRACTOR's graphs confirm EXTRACTOR's capability in capturing relevant attack behavior, to further evaluate the performance of EXTRACTOR and to measure its ability in capturing all relevant attack behaviors, we report precision, recall, and F1-score (Table IV). For this evaluation, we use the reports themselves as the ground truth and check if the activities captured in the graph are present or not in the report. We do not use the graphs generated by the experts in POIROT as ground truth, since many of those graphs contain wildcard nodes representing sets of processes. Table IV presents the performance of EXTRACTOR. As shown in Table IV, EXTRACTOR successfully captured attack behavior from the reports (with an average F-1 score 93%). However, as expected, due to language complexities, EXTRACTOR yields a small number of false positives and false negatives. Sometimes this is due to inverted edges, but more often this is due to EXTRACTOR not fully resolving some ambiguities or not detecting some entities in the text. For instance, in Uroburos pairs `credprov.tlb, load, explorer.exe` and $\star$, `fork/exec, winview.ocx` are spurious nodes and edges (see Figure 10-(e) in the Appendix). However, these did not impact threat detection (Table III).

Attack Descriptions. Figure 7 shows the graph generated by EXTRACTOR from [80], where the malware modifies several registry components and writes to several files. The divisions into left and right subgraphs in the figure reflects the report's structure, where it describes the actions performed by malware using various processes *(authorization.exe, *)*. Also, Figure 10, in the Appendix, presents other graphs generated from public CTI reports, discussed at [59], except OceanLotus [31], in which the attack behavior is described in a figure rather than natural language description.

Figure 10-(c) shows the graph generated using EXTRACTOR from the report [61]. The figure demonstrates various system calls executions with specific system entities and the asterisks processes ($\star$ inside the rectangle). The figure shows several important attacker activities and how they are connected. The graph disconnectedness is due to the writing style where the author referred to the same entity by very different names in separate sentences. For instance, the two nodes $\star$ and `8aba4b5184072f2a50cbc5ecfe326701` represent the same entity but they are separated in the graph.

Figure 10-(d), in the Appendix, shows the graph generated using EXTRACTOR from the report [7]. Due to the text complexity, EXTRACTOR generated three false positive edges *16.exec, 17.exec, and 18.exec*. Figures 10-(a), 10-(b), and 10-(e), in the Appendix, respectively, demonstrate Carbanak, HawkEye, and Uroburos graphs.

In all the cases, POIROT was able to detect the attacks, even in the presence of false positive edges.

TABLE V
CHARACTERISTICS (NODES |V| AND EDGES |E|) OF EXTRACTOR VS. MANUAL GRAPHS AND RESULTS OF THREAT DETECTION IN DARPA TC CAMPAIGNS, SCORE $P(G_1, G_2)$ AND DETECTION OUTCOME DO.

Scenario	Number of sentences	Manual		EXTRACTOR		score	Do
		|V|	|E|	|V|	|E|		
Simple APT	8	15	17	13	13	1.0	✓
Micro APT	9	13	15	15	17	0.9	✓
Drakon APT	10	10	14	14	11	0.9	✓
GatherApp	8	7	10	8	8	0.8	✓
HelloWorld	8	7	10	8	8	0.9	✓
GatherApps	8	14	14	13	12	0.8	✓
Webshell	9	7	9	12	8	0.6	✓
Metasploit	9	21	22	15	11	0.6	✓

B. *Evaluation on the DARPA Transparent Computing Dataset.*

In this experiment, we utilized the DARPA Transparent Computing campaign dataset to automatically generate the attack behavior graphs from the natural language description of the attack. During these campaigns, red-teams conducted attacks on infrastructure defended by blue teams. These attacks were carried out on four systems, including one client, one mail server, a web server, and an SSH server over a period of a couple of weeks. The text descriptions of the attacks processed by EXTRACTOR were written by the red-team members as part of the ground truth release of the exercises. These reports are shorter and more concise than those in the public CTI reports. In addition to textual descriptions, they also contain graph representations of the attacks generated by the red-team members. The graphs generated by EXTRACTOR were compared with these graphs as ground truth.

Table V describes the results of this experiment. For each attack (named in the first column), it shows the report's size in sentences (in the second column) and the manual graph's size generated by the attackers and by EXTRACTOR. The differences between the manual and EXTRACTOR's graphs are minimal due to the shorter size of the CTI reports and their conciseness. Columns seven and eight represent the results of threat hunting where POIROT [59] was used to detect the

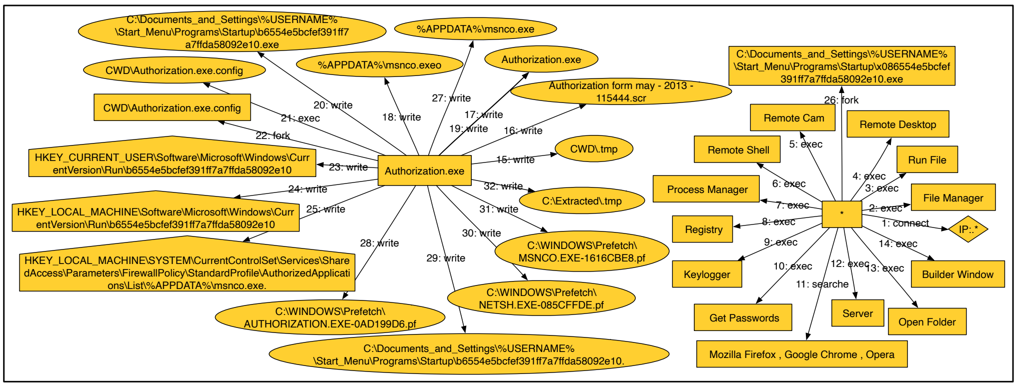

C:\Documents_and_Settings\%USERNAME%\Start_Menu\Programs\Startup\b6554e5bcfef391ff7a7ffda58092e10.exe
%APPDATA%\msnco.exe
Authorization.exe
CWD\Authorization.exe.config
%APPDATA%\msnco.exeo
Authorization form may - 2013 - 115444.scr
CWD\Authorization.exe.config
20: write
21: exec
18: write
27: write
17: write
19: write
16: write
22: fork
HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run\b6554e5bcfef391ff7a7ffda58092e10
23: write
Authorization.exe
15: write
CWD\.tmp
24: write
32: write
C:\Extracted\.tmp
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run\b6554e5bcfef391ff7a7ffda58092e10
25: write
31: write
30: write
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\SharedAccess\Parameters\FirewallPolicy\StandardProfile\AuthorizedApplications\List\%APPDATA%\msnco.exe.
28: write
29: write
C:\WINDOWS\Prefetch\MSNCO.EXE-1616CBE8.pf
C:\WINDOWS\Prefetch\NETSH.EXE-085CFFDE.pf
C:\WINDOWS\Prefetch\AUTHORIZATION.EXE-0AD199D6.pf
C:\Documents_and_Settings\%USERNAME%\Start_Menu\Programs\Startup\b6554e5bcfef391ff7a7ffda58092e10.
C:\Documents_and_Settings\%USERNAME%\Start_Menu\Programs\Startup\x086554e5bcfef391ff7a7ffda58092e10.exe
26: fork
Remote Cam
Remote Desktop
Remote Shell
5: exec
Run File
4: exec
3: exec
Process Manager
6: exec
File Manager
7: exec
2: exec
*
Registry
8: exec
1: connect
IP:.*
9: exec
14: exec
10: exec
12: exec
13: exec
Keylogger
Builder Window
11: searche
Get Passwords
Server
Open Folder
Mozilla Firefox , Google Chrome , Opera

Fig. 7. Graph generated by EXTRACTOR from njRAT [80]

TABLE VI
PRECISION, RECALL, AND F1-SCORE OF THE GRAPHS GENERATED FROM DARPA REPORTS, CALCULATED BY COMPARING EDGES IN THE AUTOMATED GRAPH AGAINST THOSE IN THE DARPA REPORTS.

Scenario	Precision	Recall	F1-Score
Simple APT	1.00	1.00	1.00
Micro APT	0.88	1.00	0.94
Drakon APT	1.00	0.84	0.91
Gather App	1.00	0.88	0.94
HelloWorld	1.00	1.00	1.00
GatherApp	1.00	1.00	1.00
Webshell	0.89	0.89	0.89
Metasploit	0.91	0.91	0.91

EXTRACTOR-generated graph inside the provenance graph generated from the audit records, which includes both attack and benign activities. In all scenarios, our detection score surpassed the detection threshold ($t \approx 0.3$), and the attack was detected successfully.

Table VI shows the performance of EXTRACTOR and its capability in capturing all relevant attack behaviors on the DARPA reports. The result shows improvement in the performance of EXTRACTOR on the DARPA CTI reports compared to the public CTI reports (Table IV). This is due to the simplicity of the DARPA reports, which resulted in generating fewer false positives. Similar to the CTI reports, most false-negatives are due to the EXTRACTOR model not being able to drive the relation from the sentence.

To further examine the possibility of generating false detection signals, we ran POIROT on a benign dataset of audit logs of the DARPA TC program, and provided in input the (attack) graphs extracted by EXTRACTOR. The dataset includes 12GB benign audit logs from the different operating systems, including Windows, Linux, FreeBSD. Our threat detection using POIROT raised no false signals. This experiment explicitly shows that EXTRACTOR graphs are concise enough not to raise false detection signals in benign environments.

C. Large Scale Experiment

To evaluate the scalability of our approach and its accuracy with additional writing styles, we process with EXTRACTOR a large number of unstructured CTI reports from two major CTI sources, namely Microsoft Security Intelligence [8] and TrendMicro Threat Encyclopedia [18].

The main challenge in this kind of evaluation is the absence of ground truth. While in the first two cases there were graphs to compare with, these CTI sources do not provide such graphs. However, the reports themselves point to a way to overcome such a challenge, described next.

Reports from these sources usually contain several sections including threat summary, technical description, and solution where they describe the overview of the attack, the technical attack details, and the steps required to remove the attack. While the first section provides some general information such as infection rate and risk and severity level about the threat, the second and the third sections provide valuable technical insight about the attack and how to reverse its impacts. Often, the last two sections of these reports are similar but antithetical to each other. In other words, while an *attack description* section describes the steps taken to compromise a system, including files created and executed, processes compromised etc, a *solution* section details the steps needed to remove the attack's artifacts, i.e., the same files created, and compromised processes. For instance, the sentence `Delete <systemfolder>\sysformat.exe from HKEY\CURRENT\USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Run` from the solution sections outlined the action required to undo the threat action described as `Adds registry value: sysformat with data: <system folder>\sysformat.exe in the registry key: HKEY\CURRENT\USER\Software\Microsoft\Windows\CurrentVersion\Run` in the *attack description* section. As another example, the sentence `check for the open connection to 10.13.13.1` from the *solution* section maps to the `connects to command and control sever 10.13.13.1` from the *attack description* section. We note, at this point, that the *solution* section does not have these characteristics across all the reports. Indeed, it often amounts to instructions on how

TABLE VII
THE REPORTS AND GRAPHS' CHARACTERISTICS AVERAGED ACROSS ALL REPORTS.

Scenario	Number of Reports	Smallest Report	Largest Report	Avg. Number of Sentences	Avg. Sentences After Summarization	Avg. Attack Behavior		Avg. Removal		Avg. MCS Score
						|V|	|E|	|V|	|E|	
Microsoft	4020	19	63	32.26	19.02	18	17	7	6	0.91
TrendMicro	11600	17	59	31.93	14.22	16	15	5	4	0.85

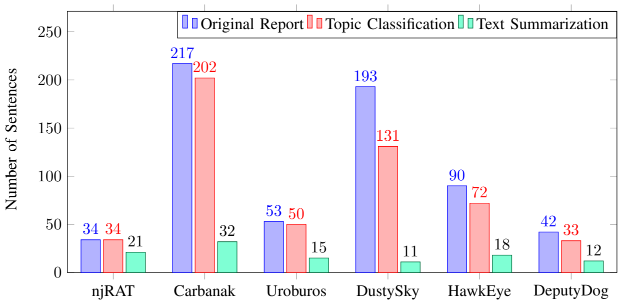


Fig. 8. The number of candidate sentences after applying text summarization, compared to the number of sentences in the raw report and the number of sentences after Topic Classification (TC).

TABLE VIII
THE PERFORMANCE EVALUATION OF CNN AND LSTM NEURAL NETWORKS VERSUS BERT LANGUAGE MODEL ON SENTENCE-VERBOSITY TASK. THE BEST RESULT ARE BOLDED.

Scenario	Precision	Recall	F-1 Score
CNN	0.895	0.895	0.897
LSTM	0.883	0.894	0.887
BERT	**0.950**	**0.957**	**0.953**

TABLE IX
EACH CELL REPRESENTS THE MCS AGAINST THE BASELINE GRAPH GENERATED BY EXTRACTOR, WITHOUT (W/O) ACTIVATING THE CORRESPONDING MODULE. ONE DENOTES RESULTS THAT ARE THE SAME AS THE BASELINE, WHILE ZERO SIGNIFIES NO MATCH BETWEEN BASELINE VERSES THE GRAPH DRAWN WITHOUT THE CORRESPONDING MODULE.

Scenario	w/o Toke.	w/o Homo.	Resolution		
			w/o ESR	w/o PR	w/o ER
njRAT	0.40	0.28	0.50	0.90	0.81
Carbanak	1.00	0.42	0.88	0.88	0.97
Uroburos	0.85	0.41	1.00	0.85	1.00
DustySky	0.71	0.0	1.00	1.00	1.00
HawkEye	0.90	0.15	0.90	0.83	1.00
DeputyDog	1.00	0.16	1.00	1.00	0.83

to download and execute a *patch* file, which does all the clean-up and patching work. However, it is relatively easy to automatically distinguish between larger *solution sections* that contain detailed clean-up steps, and smaller *solution* sections that instruct to run a patch file, and filter out the latter. The reports in [3] and [4] are examples such CTI reports, while more examples can be found at [18] and [8].

To evaluate EXTRACTOR, in this experiment, for each report that contains both an *attack description* and a detailed *solution*, we build the provenance graphs related to each section by omitting the other section from the rest of the report. Then we invert the graph obtained from the *solution* and calculate its similarity with the graph obtained from the *attack description* section. To measure the similarity between the two graphs, we use the Maximum Common Subgraph (MCS) [75], a metric that measures the containment of a smaller graph inside a larger graph. (We use this metric for this large scale evaluation as it is considerably simpler than the notion of alignment used in [59]).

Table VII shows the results of this experiment. In this table, the second column shows the number of the evaluated reports. The third and fourth columns describe the length of the smallest and largest report. The third and the fourth columns show the average number of sentences before and after text summarization. The average attack description and average solution columns show the average number of nodes and edges build from the technical details and removal section, and finally, the last column measures the similarity between the two graphs. As can be seen, the average similarity measure between the extracted graphs is equal to 0.91 for Microsoft and 0.85 for TrendMicro. This means that EXTRACTOR correctly extracts the graphs from the text in a majority of the cases.

For further comprehensiveness of this experiment, we also performed manual 'spot-checks', where we manually evaluated 50 randomly chosen reports. The individual assessments were then discussed and agreed upon in a meeting. The false-negatives and false-positives are generated due to the unresolved complexities and are still minimal, considering the complexity of the report, and are in line with our previous result. Our precision, recall and F1-score was 0.88, 0.93, 0.90.

D. Fine Grained Performance Evaluation

In this subsection, we provide more fine-grained evaluations on some of the most important steps of our approach.

Text Summarization. Previous work in automatic extraction of knowledge from CTI reports [52], [45] uses topic classification (TC) to discern irrelevant content such as advertisements from the CTI reports. In EXTRACTOR, we use a different approach for text summarization aimed at achieving a finer-grained summarization. Figure 8 shows these two approaches side by side. To do topic classification, similarly to the previous approach [52], we ran a Support Vector Machine (SVM) classifier on 1500 technical CTI sentences versus 1000 sentences of advertisement and about author details. We evaluated the model using 10-fold cross-validation, achieving a precision of 97% and a recall of 99%. As can be seen in the figure, our text summarization approach significantly reduces the size of the text compared to Topic Classification.

Finally, Table VIII presents the result of the *Sentence Verbosity* removal using the states of the art approaches, attesting that BERT outperforms other popular models.

TABLE X
THE AVERAGE NUMBER OF TIMES EACH MODULE INVOKED. TABLE SHOWS THE PREVALENCE OF CHALLENGES DISCUSSED IN SECTION II. *Homo* REFERS TO *Homogenization*.

Scenario	# of Reports	Homo.	Passive to Active	Resolution		
				ESR	PR	ER
Microsoft	4020	27.90	2.07	5.49	4.59	4.60
TrendMicro	11600	23.32	1.34	2.52	5.69	5.31

Ablation Study. To demonstrate the contribution of each EXTRACTOR module toward the final graph, we performed an ablation study to measure the similarity of the graph generated in the absence of that module compared to the baseline (having all modules active).

Table IX shows results of our ablation study. Each column represents the result of the MCS similarity score of EXTRACTOR's generated graph in the absence of that component(w/o). The baseline for this study is the overall performance of EXTRACTOR, set at 1. Each column shows the loss of performance when any specific component is omitted in the overall approach. The table also shows the diversity in the writing styles and the fact that every single technique matters (as there is no column with all 1's), showing the need to combine these different techniques across the various reporting scenarios. Our results show that all the different modules in EXTRACTOR successfully contribute to various degrees, depending on the text style, to the concise graph generation. In particular, they enable EXTRACTOR to process a wide variety of writing styles successfully.

In addition, we examined the impact of the steps of Normalization, Resolution, and Summarization, by not performing these steps and rather building the graphs from the raw reports. Figure 9 shows that the size of such graphs is in the hundreds of edges, while the size of the graphs obtained by the full chain of modules is much smaller.

Table X shows the prevalence of challenges discussed in Section II in two of the major threat report websites [8] and [18]. Each column represents the number of times that each module has been invoked (we avoid adding tokenizer as it has been reflected in Table VII). The second column shows the total number of analyzed reports. The third and the fourth columns outline the average number of homogenized instances and passive to active conversation, respectively. Finally, the resolution column presents the results of ESR, PR, and ER.

Table XI presents the SRL's performance before and after retraining. Finally, Table XII shows the performance of the SEE module in picking correct and meaningful nodes. To evaluate our SEE's completeness, we ran our SEE module on 1,000 public reports used by [89] and compared SSE results against their result as a baseline. Table XII presents the result of this evaluation.

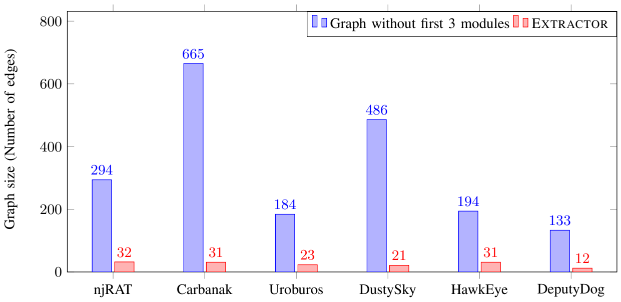


Fig. 9. The size of the generated graph before and after utilizing Normalization, Resolution, and Summarization modules.

TABLE XI
THE PERFORMANCE EVALUATION BEFORE AND AFTER TRAINING THE SRL MODEL ON CTI DATA, WHERE WE USED 80%, 20%, 20% FOR THE TRAIN, TEST, AND VALIDATION, RESPECTIVELY.

Scenario	Precision	Recall	F-1 Score
SRL(Base)	0.83	0.82	0.84
SRL(Retrained)	0.92	0.93	0.93

TABLE XII
THE PERFORMANCE OF SEE MODULE.

Scenario	Precision	Recall	F-1 Score
SEE	1	0.98	0.99

VI. DISCUSSION AND LIMITATIONS

False Positives and False Negatives. As shown in Tables IV and VI there are extraneous nodes or edges in the EXTRACTOR generated graphs. As discussed in Sections II and III, we note that some loss of precision in extraction is inevitable due to general issues in dealing with natural language complexity. What is, therefore, the significance of these extraneous nodes and edges (or missing ones) in the EXTRACTOR generated graph with respect to the 'big picture' of the threat hunting problem? To answer this, we note our choice of a threat hunting approach that uses approximate matching [59], facilitates us to successfully identify the threat despite the extraneous information. In fact, we can go on to argue that such approximate techniques are required of the general threat-hunting problem, as it is likely that not all of the activities described in a threat report are likely to manifest in a host due to intrinsic factors (e.g., the non-deterministic factors such as randomness or time affecting the execution of a threat binary) in the activity-based provenance graphs. Therefore, the approximation that is inherently needed for threat-hunting is able to work with the small loss of precision inherent to NLP, as shown in Section V.

Limitations and Future Works. EXTRACTOR' s performance may diminish in scenarios where the description of an action spans several sentences or a paragraph, where subject or object resolution might face challenges based on how the threat report was authored. As an instance of this challenge, see the discussion about Figure 10-(c) in Section V-A. EXTRACTOR may deal with this issue if additional information in the form of alternate reports is made available to resolve these challenges. Another way to resolve this is to facilitate mechanisms in EXTRACTOR to actively collaborate with the human operator to resolve these entities.

Another limitation of our graph-based approach is that it is not applicable to attacks that involve timing, side-channel inference, etc. The graphs describing the attack behavior are modelled after audit logs that do not capture information at a granularity that enables these attacks to be detected. However, this limitation is common with other approaches that involve provenance graph-based detection or threat hunting. Similarly, our approach only extracts the attack behaviors described in the natural language and cannot resolve the behaviors represented in other forms like figures and charts.

Various modules of EXTRACTOR use dictionaries to enhance the overall performance. While we have tried to be complete in choosing appropriate words, there may be reports where people use words that are not in the dictionaries. Therefore, there can be room to improve the dictionaries further. Using the Named Entity Recognition (NER) system may also enhance the approach in translating noun phrases into unified system representations. Moreover, future works may extend the EXTRACTOR to infer the graph from unstructured vulnerability reports. These graphs can be further used to detect possible vulnerabilities within the application.

VII. RELATED WORK

Provenance Graph Analytics. Several research projects utilize system audit logs for attack reconstruction and forensic analysis, and threat hunting [42], [36], [37], [59], [53], [73]. Hercule [69] rebuilds attack stages through comparing logs collected from various sources. Bilge et al. [23] leverage NetFlow logs for detecting botnet C&C channels. Oprea et al. [66] uses web proxy and DNS logs to identify infections in enterprise networks.

King et al. [49], [48] introduced the practice of constructing provenance graphs from kernel audit logs. Several studies have used provenance graphs in attack detection and forensics [43], [51], [85], [38]. Hossain et al. [43], Xu et al. [85], and LogGC [51] proposed reduction techniques that reduce the size of the graph while maintaining the accuracy needed for forensic analysis. EXTRACTOR can be a companion to these approaches to provide a clear picture of attacks.

NLP and Threat Information Extraction. Several open standards such as STIX [15], MISP [72], and OpenIOC [11] have been proposed to exchange knowledge about IOCs in an interoperable way. However, unlike our approach, these standards are more focused on exchanging IOCs than describing how those IOCs are connected and how the attacks behave (see the examples at [12]). Companies that use threat exchange standards instead of only relying on the exchange of high-level threat data can benefit from the publicly available knowledge in the wild. On the other hand, these threat exchange standards are limited in usage, as companies are not equally interested in sharing their data. Moreover, the exchanged data does not contain technical details such as the affected registry, file path, and the application name as they can be a privacy leakage of the company's private information. As in many cases, the organizations' internal policies prevent the sharing of data with outside entities [77], [26].

The VirusTotal Graph [21] also differs from our work as it only represents the high-level view of the attack, mainly including hashes, IPs, and domains involved in a possible threat scenario. Also, unlike our approach, VirusTotal's report is generated based on the analysis of sample malware, while EXTRACTOR by having access to the publicly available reports (which can include the VirusTotal) allows utilization of public CTI, converting raw reports into actionable knowledge.

iACE [52] proposes a graph mining approach to extract IOCs from security articles. ChainSmith [89] uses NLP to extract IOCs from security articles and further categorize them into campaign stages. TTPDrill [45] proposes an ontology which helps to understand the characteristics and specifications of cyber threats. It uses NLP and Information retrieval (IR) to extract the threat actions from reports. The work of [46] creates TTP chains from reports, using DP rules. Unlike these approaches, EXTRACTOR focuses on extracting the attack behavior and captures system-level causality in the form of the provenance graphs.

SemFuzz [86] performs fuzzing guided by information extraced from vulnerability reports. Feng et al. [32] use NLP to generate network signatures from unstructured vulnerability reports. They use those signatures in intrusion detection and firewall systems. Dong et al. [29] use Named Entity Recognition and Relation Extraction to extract software name and version and report inconsistency between major vulnerability databases. Even though, somehow related, EXTRACTOR's goal and techniques are essentially different from these works.

Featuresmith [88] generates a feature set for detecting Android malware from security literature. In contrast, EXTRACTOR aims to build a provenance graph that represents the actual behavior of the attack. Privee [90] leverages machine learning to retrieve web policies. The work of [74] and [68] relate the app description with permissions using NLP. The works of [63] and [44] identify users' sensitive inputs in Android app. EKLAVYA [25] uses NLP to recover function signatures from binary code.

VIII. CONCLUSION

EXTRACTOR automatically builds a provenance graph from CTI reports written in natural language. We evaluate EXTRACTOR using various threat reports and real-world attack scenarios. EXTRACTOR successfully extracts graphs that match those drawn manually by security experts, and those graphs were successfully used for threat detection.

IX. ACKNOWLEDGMENTS

We would like to thank reviewers for their helpful review comments and suggestions to the manuscript. Thanks are especially due to our anonymous shepherd who provided many useful suggestions for refinement. This work was supported by DARPA under SPAWAR (N6600118C4035) and NSF (CNS-1918542). The views, opinions, and/or findings expressed are those of the authors and should not be interpreted as representing the official views or policies of the U.S. Government.

References

[1] "word2vec," 2013, available at: https://code.google.com/archive/p/word2vec/.

[2] "GloVe: Global Vectors for Word Representation," 2014, https://nlp.stanford.edu/projects/glove/.

[3] "Bkdr_kuluoz.en," 2015, available at:https://www.trendmicro.com/vinfo/us/threat-encyclopedia/malware/bkdr_kuluoz.en.

[4] "Backdoor:win32/rbot.fg," 2017, available at:https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Backdoor:Win32/Rbot.FG&threatId=68811.

[5] "Sentence length has declined 75% in the past 500 years," 2017, available at:https://medium.com/@theacropolitan/sentence-length-has-declined-75-in-the-past-500-years-2e40f80f589f.

[6] "Apt and cybercriminals campaign collection," 2019, available at:https://github.com/CyberMonitor/APT_CyberCriminal_Campagin_Collections.

[7] "Operation dustysky," 2019, available at:https://www.clearskysec.com/wp-content/uploads/2016/01/OperationDustySky_TLP_WHITE.pdf.

[8] "Global threat activity / microsoft security intelligence," 2020, available at:https://www.microsoft.com/en-us/wdsi/threats.

[9] "Industrial-strength natural language processing," 2020, available at:https://spacy.io/.

[10] "Neuralcoref 4.0: Coreference resolution in spacy with neural networks." 2020, available at:https://github.com/huggingface/neuralcoref.

[11] "OpenIOC," 2020, available at: http://openIOC.org.

[12] "Sample STIX reports," 2020, https://stixproject.github.io/examples/.

[13] "Stanford corenlp – natural language software," 2020, available at:https://stanfordnlp.github.io/CoreNLP/.

[14] "Stanford tokenizer," 2020, available at: https://nlp.stanford.edu/software/tokenizer.shtml.

[15] "Structured threat information expression (stix) 1.x archive website," 2020, available at:https://stixproject.github.io/.

[16] "Symantec security center," 2020, available at:https://www.broadcom.com/support/security-center.

[17] "Synonyms," 2020, https://www.thesaurus.com.

[18] "Threat encyclopedia," 2020, available at:https://www.trendmicro.com/vinfo/us/threat-encyclopedia/.

[19] "Transparent Computing," 2020, https://www.darpa.mil/program/transparent-computing.

[20] "Virus Radar Threat Encyclopaedia," 2020, https://www.virusradar.com/en.

[21] "VT Graph," 2020, https://www.virustotal.com/gui/graph-overview.

[22] E. AI, "Annotation specifications," 2020, available at:https://spacy.io/api/annotation.

[23] L. Bilge, D. Balzarotti, W. Robertson, E. Kirda, and C. Kruegel, "Disclosure: detecting botnet command and control servers through large-scale netflow analysis," in *Proceedings of the 28th Annual Computer Security Applications Conference*, 2012, pp. 129–138.

[24] C. Chen and V. Ng, "Chinese zero pronoun resolution with deep neural networks," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2016, pp. 778–788.

[25] Z. L. Chua, S. Shen, P. Saxena, and Z. Liang, "Neural nets can learn function type signatures from binaries," in *26th {USENIX} Security Symposium ({USENIX} Security 17)*, 2017, pp. 99–116.

[26] J. M. de Fuentes, L. González-Manzano, J. Tapiador, and P. Peris-Lopez, "Pracis: Privacy-preserving and aggregatable cybersecurity information sharing," *computers & security*, vol. 69, pp. 127–141, 2017.

[27] M.-C. De Marneffe and C. D. Manning, "Stanford typed dependencies manual," Technical report, Stanford University, Tech. Rep., 2008.

[28] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," *CoRR*, vol. abs/1810.04805, 2018. [Online]. Available: http://arxiv.org/abs/1810.04805

[29] 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)*, 2019, pp. 869–885.

[30] S. K. D'Mello, "On the influence of an iterative affect annotation approach on inter-observer and self-observer reliability," *IEEE Transactions on Affective Computing*, vol. 7, no. 2, pp. 136–149, 2015.

[31] ESET, "Oceanlotus old techniques, new backdoor," 2018, available at:https://www.welivesecurity.com/wp-content/uploads/2018/03/ESET_OceanLotus.pdf.

[32] X. Feng, X. Liao, X. Wang, H. Wang, Q. Li, K. Yang, H. Zhu, and L. Sun, "Understanding and securing device vulnerabilities through automated bug report analysis," in *SEC'19: Proceedings of the 28th USENIX Conference on Security Symposium*, 2019.

[33] K. Fort, *Collaborative Annotation for Reliable Natural Language Processing: Technical and Sociological Aspects*. John Wiley & Sons, 2016.

[34] M. Gardner, J. Grus, M. Neumann, O. Tafjord, P. Dasigi, N. F. Liu, M. E. Peters, M. Schmitz, and L. Zettlemoyer, "Allennlp: A deep semantic natural language processing platform," *CoRR*, vol. abs/1803.07640, 2018. [Online]. Available: http://arxiv.org/abs/1803.07640

[35] Gdata, "The uroburos case: new sophisticated rat identified," 2015, available at:https://www.gdatasoftware.com/blog/2014/11/23937-the-uroburos-case-new-sophisticated-rat-identified.

[36] A. Goel, W.-C. Feng, D. Maier, and J. Walpole, "Forensix: A robust, high-performance reconstruction system," in *25th IEEE International Conference on Distributed computing systems workshops*. IEEE, 2005, pp. 155–162.

[37] A. Goel, K. Po, K. Farhadi, Z. Li, and E. De Lara, "The taser intrusion recovery system," in *Proceedings of the twentieth ACM symposium on Operating systems principles*, 2005, pp. 163–176.

[38] W. U. Hassan, L. Aguse, N. Aguse, A. Bates, and T. Moyer, "Towards scalable cluster auditing through grammatical inference over provenance graphs," in *Network and Distributed Systems Security Symposium*, 2018.

[39] W. U. Hassan, S. Guo, D. Li, Z. Chen, K. Jee, Z. Li, and A. Bates, "Nodoze: Combatting threat alert fatigue with automated provenance triage." in *NDSS*, 2019.

[40] L. He, K. Lee, M. Lewis, and L. Zettlemoyer, "Deep semantic role labeling: What works and what's next," in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2017, pp. 473–483.

[41] M. N. Hossain, S. M. Milajerdi, J. Wang, B. Eshete, R. Gjomemo, R. Sekar, S. Stoller, and V. Venkatakrishnan, "{SLEUTH}: Real-time attack scenario reconstruction from {COTS} audit data," in *26th {USENIX} Security Symposium ({USENIX} Security 17)*, 2017, pp. 487–504.

[42] M. N. Hossain, S. Sheikhi, and R. Sekar, "Combating dependence explosion in forensic analysis using alternative tag propagation semantics," in *2020 IEEE Symposium on Security and Privacy (S&P)*. IEEE, 2020.

[43] M. N. Hossain, J. Wang, O. Weisse, R. Sekar, D. Genkin, B. He, S. D. Stoller, G. Fang, F. Piessens, E. Downing *et al.*, "Dependence-preserving data compaction for scalable forensic analysis," in *27th {USENIX} Security Symposium ({USENIX} Security 18)*, 2018, pp. 1723–1740.

[44] J. Huang, Z. Li, X. Xiao, Z. Wu, K. Lu, X. Zhang, and G. Jiang, "{SUPOR}: Precise and scalable sensitive user input detection for android apps," in *24th {USENIX} Security Symposium ({USENIX} Security 15)*, 2015, pp. 977–992.

[45] 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*. ACM, 2017, pp. 103–115.

[46] G. Husari, E. Al-Shaer, B. Chu, and R. F. Rahman, "Learning apt chains from cyber threat intelligence," in *Proceedings of the 6th Annual Symposium on Hot Topics in the Science of Security*, 2019, pp. 1–2.

[47] KasperSky, "Carbanak apt the great bank robbery," 2015, available at:https://media.kasperskycontenthub.com/wp-content/uploads/sites/43/2018/03/08064518/Carbanak_APT_eng.pdf.

[48] S. T. King and P. M. Chen, "Backtracking intrusions," in *Proceedings of the nineteenth ACM symposium on Operating systems principles*, 2003, pp. 223–236.

[49] S. T. King, Z. M. Mao, D. G. Lucchetti, and P. M. Chen, "Enriching intrusion alerts through multi-host causality." in *NDSS*, 2005.

[50] Y. Kwon, F. Wang, W. Wang, K. H. Lee, W.-C. Lee, S. Ma, X. Zhang, D. Xu, S. Jha, G. F. Ciocarlie *et al.*, "Mci: Modeling-based causality inference in audit logging for attack investigation." in *NDSS*, 2018.

[51] K. H. Lee, X. Zhang, and D. Xu, "Loggc: garbage collecting audit log," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, 2013, pp. 1005–1016.

[52] 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*. ACM, 2016, pp. 755–766.

[53] Y. Liu, M. Zhang, D. Li, K. Jee, Z. Li, Z. Wu, J. Rhee, and P. Mittal, "Towards a timely causality analysis for enterprise security." in *NDSS*, 2018.

[54] E. Loper and S. Bird, "NLTK: the natural language toolkit," *CoRR*, vol. cs.CL/0205028, 2002. [Online]. Available: https://arxiv.org/abs/cs/0205028

[55] S. Ma, K. H. Lee, C. H. Kim, J. Rhee, X. Zhang, and D. Xu, "Accurate, low cost and instrumentation-free security audit logging for windows," in *Proceedings of the 31st Annual Computer Security Applications Conference*, 2015, pp. 401–410.

[56] S. Ma, X. Zhang, and D. Xu, "Protracer: Towards practical provenance tracing by alternating between logging and tainting." in *NDSS*, 2016.

[57] McClosky, Charniak, and Johnson, "Automatic domain adaptation for parsing," in *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, 2010, pp. 28–36.

[58] D. McClosky, E. Charniak, and M. Johnson, "Reranking and self-training for parser adaptation," in *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2006, pp. 337–344.

[59] S. M. Milajerdi, B. Eshete, R. Gjomemo, and V. Venkatakrishnan, "Poirot: Aligning attack behavior with kernel audit records for cyber threat hunting," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 1795–1812.

[60] G. A. Miller, "Wordnet: a lexical database for english," *Communications of the ACM*, vol. 38, no. 11, pp. 39–41, 1995.

[61] N. Moran and V. Nart, "Analysis of a new hawkeye variant," 2013, available at:https://www.fireeye.com/blog/threat-research/2013/09/operation-deputydog-zero-day-cve-2013-3893-attack-against-japanese-targets.html.

[62] D. Mu, A. Cuevas, L. Yang, H. Hu, X. Xing, B. Mao, and G. Wang, "Understanding the reproducibility of crowd-reported security vulnerabilities," in *27th {USENIX} Security Symposium ({USENIX} Security 18)*, 2018, pp. 919–936.

[63] Y. Nan, M. Yang, Z. Yang, S. Zhou, G. Gu, and X. Wang, "Uipicker: User-input privacy identification in mobile applications," in *24th {USENIX} Security Symposium ({USENIX} Security 15)*, 2015, pp. 993–1008.

[64] S. Nariyama, "Subject ellipsis in english," *Journal of Pragmatics*, vol. 36, no. 2, pp. 237–264, 2004.

[65] V. Ng and C. Cardie, "Improving machine learning approaches to coreference resolution," in *Proceedings of the 40th annual meeting on association for computational linguistics*. Association for Computational Linguistics, 2002, pp. 104–111.

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

[67] M. Palmer, D. Gildea, and P. Kingsbury, "The proposition bank: An annotated corpus of semantic roles," *Computational linguistics*, vol. 31, no. 1, pp. 71–106, 2005.

[68] R. Pandita, X. Xiao, W. Yang, W. Enck, and T. Xie, "{WHYPER}: Towards automating risk assessment of mobile applications," in *Presented as part of the 22nd {USENIX} Security Symposium ({USENIX} Security 13)*, 2013, pp. 527–542.

[69] K. Pei, Z. Gu, B. Saltaformaggio, S. Ma, F. Wang, Z. Zhang, L. Si, X. Zhang, and D. Xu, "Hercule: Attack story reconstruction via community discovery on correlated log graph," in *Proceedings of the 32Nd Annual Conference on Computer Security Applications*, 2016, pp. 583–595.

[70] J. W. Pennebaker, "The secret life of pronouns," *New Scientist*, vol. 211, no. 2828, pp. 42–45, 2011.

[71] M. Pfaff and H. Krcmar, "Natural language processing techniques for document classification in it benchmarking," in *Proceedings of the 17th International Conference on Enterprise Information Systems-Volume 1*. SCITEPRESS-Science and Technology Publications, Lda, 2015, pp. 360–366.

[72] M.-O. S. T. I. Platform, "Open standards for threat information sharing," 2020, available at:http://www.misp-project.org/index.html.

[73] D. J. Pohly, S. McLaughlin, P. McDaniel, and K. Butler, "Hi-fi: collecting high-fidelity whole-system provenance," in *Proceedings of the 28th Annual Computer Security Applications Conference*, 2012, pp. 259–268.

[74] Z. Qu, V. Rastogi, X. Zhang, Y. Chen, T. Zhu, and Z. Chen, "Autocog: Measuring the description-to-permission fidelity in android applications," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, 2014, pp. 1354–1365.

[75] J. W. Raymond and P. Willett, "Maximum common subgraph isomorphism algorithms for the matching of chemical structures," *Journal of computer-aided molecular design*, vol. 16, no. 7, pp. 521–533, 2002.

[76] E. Riloff *et al.*, "Automatically constructing a dictionary for information extraction tasks," in *AAAI*, vol. 1, no. 1. Citeseer, 1993, pp. 2–1.

[77] F. Sadique, S. Cheung, I. Vakilinia, S. Badsha, and S. Sengupta, "Automated structured threat information expression (stix) document generation with privacy preservation," in *2018 9th IEEE Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*. IEEE, 2018, pp. 847–853.

[78] J. Shang, L. Liu, X. Ren, X. Gu, T. Ren, and J. Han, "Learning named entity tagger using domain-specific dictionary," *CoRR*, vol. abs/1809.03599, 2018. [Online]. Available: http://arxiv.org/abs/1809.03599

[79] P. Shi and J. Lin, "Simple BERT models for relation extraction and semantic role labeling," *CoRR*, vol. abs/1904.05255, 2019. [Online]. Available: http://arxiv.org/abs/1904.05255

[80] F. C. Solutions, "njrat," 2013, available at:https://app.box.com/s/vdg51zbfvap52w60zj0is3l1dmyya0n4.

[81] W. M. Soon, H. T. Ng, and D. C. Y. Lim, "A machine learning approach to coreference resolution of noun phrases," *Computational linguistics*, vol. 27, no. 4, pp. 521–544, 2001.

[82] spaCy, "en_core_web_lg," 2020, available at:https://spacy.io/models/en#en_core_web_lg.

[83] R. Sukthanker, S. Poria, E. Cambria, and R. Thirunavukarasu, "Anaphora and coreference resolution: A review," *CoRR*, vol. abs/1805.11824, 2018. [Online]. Available: http://arxiv.org/abs/1805.11824

[84] X. Sun, J. Dai, P. Liu, A. Singhal, and J. Yen, "Using bayesian networks for probabilistic identification of zero-day attack paths," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 10, pp. 2506–2521, 2018.

[85] Z. Xu, Z. Wu, Z. Li, K. Jee, J. Rhee, X. Xiao, F. Xu, H. Wang, and G. Jiang, "High fidelity data reduction for big data security dependency analyses," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 504–516.

[86] W. You, P. Zong, K. Chen, X. Wang, X. Liao, P. Bian, and B. Liang, "Semfuzz: Semantics-based automatic generation of proof-of-concept exploits," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 2139–2154.

[87] X. Zhang, "Analysis of a new hawkeye variant," 2019, available at:https://www.fortinet.com/blog/threat-research/hawkeye-malware-analysis.html.

[88] Z. Zhu and T. Dumitraş, "Featuresmith: Automatically engineering features for malware detection by mining the security literature," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 767–778.

[89] Z. Zhu and T. Dumitras, "Chainsmith: Automatically learning the semantics of malicious campaigns by mining threat intelligence reports," in *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2018, pp. 458–472.

[90] S. Zimmeck and S. M. Bellovin, "Privee: An architecture for automatically analyzing web privacy policies," in *23rd {USENIX} Security Symposium ({USENIX} Security 14)*, 2014, pp. 1–16.

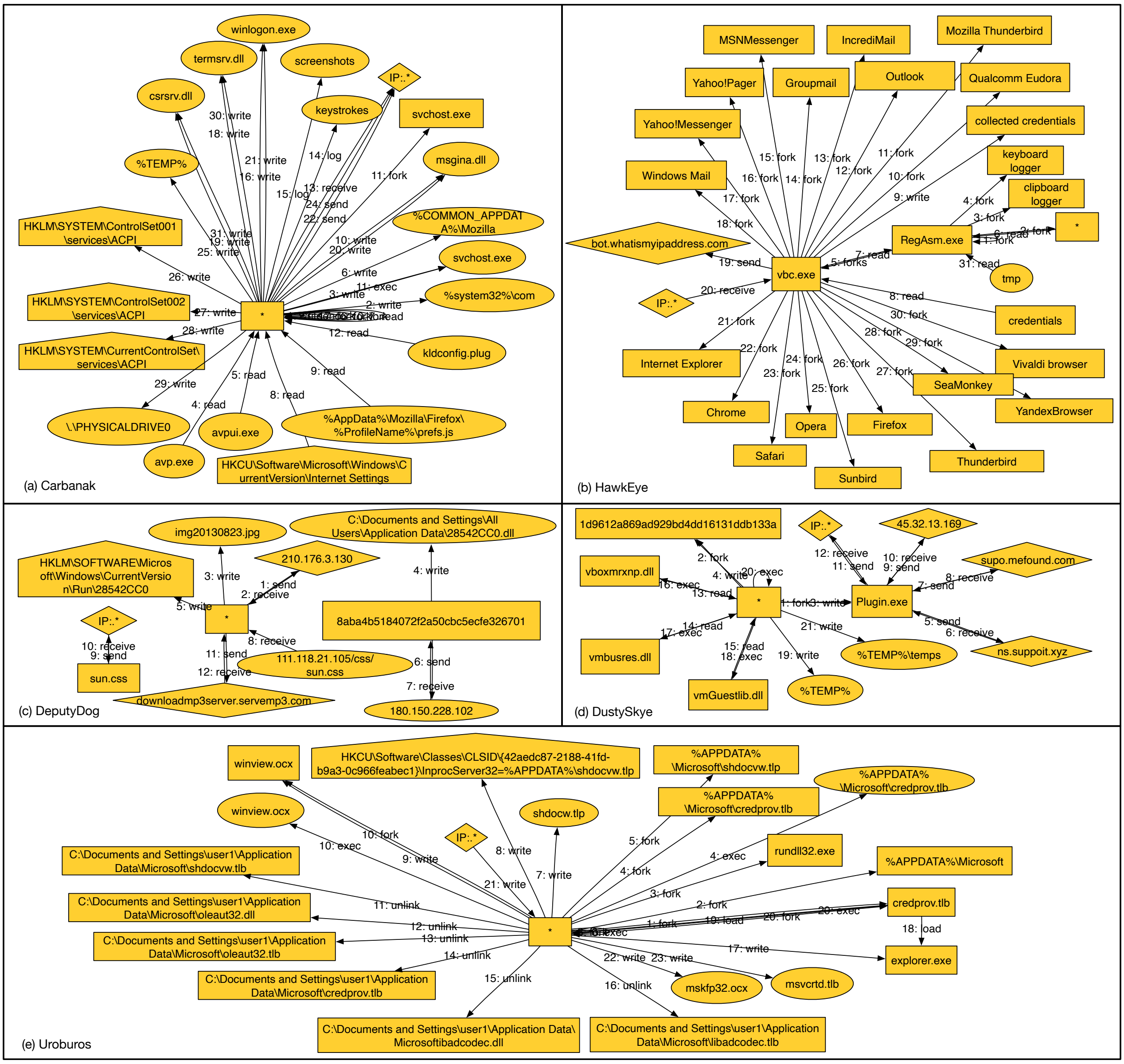
X. Appendix

TABLE XIII
SAMPLE OF NOUN PHRASES AND THEIR CORRESPONDING SYNONYMS IN NOUN DICTIONARY.

Noun	Synonyms
IP:.*	'CC server', 'CC', 'command and control sever', 'C2 server', 'c2', 'candc server', 'C2 node', 'CandC', 'CandC', 'command controle sever', 'C2', 'CandC server', 'CC server', 'CommandControle sever', 'Command Controle',...
TEMP	'%TEMP%, '<TEMP> ', 'Windows temporary folder', 'temporary folder' , '%Temporary folder%, 'TMP', '%Temp Folder%' '%Temp directory%,...
Home Folder	'%HOMEPATH%', '<HOMEPATH>', '%HOME_PATH%', '<HOME_PATH>', '%HOME%', '<HOME folder>', <HOME Directory>', 'USER PATH', '%USER Directory%',...

TABLE XIV
MALWARE REPORTS DETAILS AND CHARACTERISTICS.

Scenario	Year	Risk	Submitted Samples	Primary target	Malware MD5	Description
njRAT [80] - fig. 7	2013	High	30	Middle eastern governments, energy sectors, and telecom industries.	2013385034e5c8dfbbe47958fd821ca0	The malware has several capabilities, including logging keystrokes, uploading and downloading files, recording the victim's camera, steal user credentials stored in the system, open a reverse shell, and manipulations of the process, file, and registry, etc.
Carbanak [47] - fig. 10-(a)	2015	High	109	Banking and financial institutions	1e47e12d11580e935878b0ed78d2294f	The malware has several capabilities, including logging keystrokes, uploading and downloading files, recording the victim's camera, steal user credentials stored in the system, open a reverse shell, and manipulations of the process, file, and registry, etc.
Uroburos [35] - fig. 10-(e)	2014	High	4	"the most significant breach of U.S. military computers"[35]	51e7e58a1e654b6e586fe36e10c67a73	The malware exploits vulnerabilities in Java (CVE-2012-1723), Adobe Flash (unknown) or Internet Explorer 6, 7, 8 exploits (unknown), and is capable of performing a wide range of tasks.
DustySky [7] - fig. 10-(d)	2015	High	79	Intelligence gathering with Political motives	0756357497c2cd7f41ed6a6d4403b395	The malware is written in .NET by a politically-motivated group with primary targets in the Middle East, Europe, and the United States and can collect a wide range of details from the target system.
HawkEye [87] - fig. 10-(b)	2019	High	3	A wide range of industries and sectors	666a200148559e4a83fabb7a1bf655ac	The malware has several capabilities, including stealing email credentials, logging keystrokes, taking screenshots, USB propagation, stealing Bitcoin wallet info, Antivirus, firewall checking, etc.
DeputyDog [61] - fig. 10-(c)	2013	High	8	Against Japanese Targets	8aba4b5184072f2a50cbc5ecfe326701	ZeroDay CVE-2013-3893 against Microsoft internet explorer - Japan



(a) Carbanak
winlogon.exe
termsrv.dll
screenshots
IP:.*
csrsrv.dll
keystrokes
svchost.exe
msgina.dll
%TEMP%
HKLM\SYSTEM\ControlSet001\services\ACPI
%COMMON_APPDATA%\Mozilla
svchost.exe
HKLM\SYSTEM\ControlSet002\services\ACPI
%system32%\com
*
HKLM\SYSTEM\CurrentControlSet\services\ACPI
kldconfig.plug
\\.\PHYSICALDRIVE0
avpui.exe
%AppData%\Mozilla\Firefox\%ProfileName%\prefs.js
avp.exe
HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings
(b) HawkEye
MSNMessenger
IncrediMail
Mozilla Thunderbird
Yahoo!Pager
Groupmail
Outlook
Qualcomm Eudora
Yahoo!Messenger
collected credentials
Windows Mail
keyboard logger
clipboard logger
bot.whatismyipaddress.com
RegAsm.exe
*
vbc.exe
tmp
IP:.*
credentials
Internet Explorer
Vivaldi browser
SeaMonkey
YandexBrowser
Chrome
Opera
Firefox
Safari
Sunbird
Thunderbird
(c) DeputyDog
img20130823.jpg
C:\Documents and Settings\All Users\Application Data\28542CC0.dll
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\28542CC0
210.176.3.130
IP:.*
*
8aba4b5184072f2a50cbc5ecfe326701
111.118.21.105/css/sun.css
sun.css
downloadmp3server.servemp3.com
180.150.228.102
(d) DustySkye
1d9612a869ad929bd4dd16131ddb133a
IP:.*
45.32.13.169
vboxmrxnp.dll
supo.mefound.com
*
Plugin.exe
vmbusres.dll
%TEMP%\temps
ns.suppoit.xyz
vmGuestlib.dll
%TEMP%
(e) Uroburos
winview.ocx
HKCU\Software\Classes\CLSID\{42aedc87-2188-41fd-b9a3-0c966feabec1}\InprocServer32=%APPDATA%\shdocvw.tlp
%APPDATA%\Microsoft\shdocvw.tlp
%APPDATA%\Microsoft\credprov.tlb
winview.ocx
shdocw.tlp
%APPDATA%\Microsoft\credprov.tlb
C:\Documents and Settings\user1\Application Data\Microsoft\shdocvw.tlb
IP:.*
rundll32.exe
%APPDATA%\Microsoft
C:\Documents and Settings\user1\Application Data\Microsoft\oleaut32.dll
credprov.tlb
*
C:\Documents and Settings\user1\Application Data\Microsoft\oleaut32.tlb
explorer.exe
C:\Documents and Settings\user1\Application Data\Microsoft\credprov.tlb
mskfp32.ocx
msvcrtd.tlb
C:\Documents and Settings\user1\Application Data\Microsoftibadcodec.dll
C:\Documents and Settings\user1\Application Data\Microsoft\libadcodec.tlb

Fig. 10. Graphs after generalization, keeping IOCs and asterisking unknown system entities.