# Automatic Generation of Malware Threat Intelligence from Unstructured Malware Traces

Yuheng Wei and Futai Zou(✉)

School of Cyber Science and Engineering, Shanghai Jiao Tong University, Shanghai, China
{geralweiyh,zoufutai}@sjtu.edu.cn

**Abstract.** Sharing plenty and accurate structured Cyber Threat Intelligence (CTI) will play a pivotal role in adapting to rapidly evolving cyber attacks and malware. However, the traditional CTI generation methods are extremely time and labor-consuming. The recent work focuses on extracting CTI from well structured Open Source Intelligence (OSINT). However, many challenges are still to generate CTI and Indicators of Compromise(IoC) from non-human-written malware traces. This work introduces a method to automatically generate concise, accurate and understandable CTI from unstructured malware traces. For a specific class of malware, we first construct the IoC expressions set from malware traces. Furthermore, we combine the generated IoC expressions and other meaningful information in malware traces to organize the threat intelligence which meets open standards such as Structured Threat Information Expression (STIX). We evaluate our algorithm on real-world dataset. The experimental results show that our method achieves a high average recall rate of 89.4% on the dataset and successfully generates STIX reports for every class of malware, which means our methodology is practical enough to automatically generate effective IoC and CTI.

**Keywords:** Malware · Threat intelligence · Indicators of compromise

## 1 Introduction

Malware (short for malicious software) has been used as a weapon by the threat actors. Many types of malware, including computer virus, trojans, worms, ransomware, rootkit, and bots, are now active on the Internet, posing threats to Internet users [30]. We can see the crazy malware number growth year by year, referring to many security company's annual reports. Unfortunately, it is hard to win the security war with the rapidly growing and evolving malware using traditional malware analysis methods.

Plenty and accurate structured Cyber Threat Intelligence (CTI) will play a pivotal role in adapting to the rapidly evolving cyber attacks and malware [13]. However, the traditional CTI generation method is extremely time and labor-consuming. Various techniques have been developed for improving malware

J. Garcia-Alfaro et al. (Eds.): SecureComm 2021, LNICST 398, pp. 44–61, 2021.
https://doi.org/10.1007/978-3-030-90019-9_3

detection algorithms and methods, but not for reporting. Traditional CTI generation methods rely on security experts to summarize the massive, easy-to-get but low-value basic data, such as Hash, network traces, and host information. Such an inefficient method is impossible to deal with the rapid development and growth of malware. Security industries or organizations may have collected a large number of malware samples from the Internet; what they need is an effective and robust method to automatically summarize the features of these samples and then generate usable CTI. To address this issue, we need to find a reliable technology or system to analyze threat information and generate structured threat intelligence automatically.

There is extensive research on machine learning based CTI generation methods, especially IoC generation methods based on artificial intelligence algorithms. Many of these machine learning based methods use online security articles or malware analysis results, for example, Symantec's malware or APT reports, as their information sources. Under the intuition that some information that can be used to generate a structured CTI will be presented in a similar way in these articles or malware reports, researchers try to use artificial intelligence algorithms like NLP to extract this useful information. Of course, these machine learning based methods can generate high-quality IoC, however, they only cover well-organized malware analysis reports, which means these are not available methods for those security industries and organizations that only have a large number of raw malware samples. When we try to use machine learning methods like decision tree to generate IoC from malware static analysis and dynamic analysis results, they may generate too complex signatures, and it is hard to use in real malware detection.

This work will introduce a practical methodology to automatically generate concise, accurate, and understandable CTI from unstructured malware traces. Using malware samples collected from the Internet, we obtain malware traces through sandbox analysis. Since IoC is one of the most important parts of CTI, we first propose a method to generate IoC from malware traces. Furthermore, we combine the generated IoC and other meaningful strings in malware traces, such as malware class information, attacker's IP address, and observed intrusion action, to organize a CTI meeting certain standards.

**Contributions.** Our contributions are as follows:

- We develop a practical and easy-to-deploy methodology to generate structured CTI from raw malware traces automatically. We first get traces of malware samples by sandbox analysis, then generate accurate and concise IoC after data preprocessing. We finally integrate generated IoC and other meaningful information in malware traces to generate well-structured CTI.
- We study the malware sandbox analysis report structure and design several rules for removing low-value strings from it. Usually, the malware traces consist of static analysis and dynamic analysis results, but not all of the strings in the reports help generate IoC and CTI. We design rules to remove meaningless strings in malware traces and only select those features that can represent characteristics well.

– We propose an effective Greedy IoC Generation algorithm, *GIG* for short, to generate concise, accurate, and understandable IoC from malware traces. For a specific class of malware, we first add up all the possible IoC candidates and merge similar expressions into one regular expression based IoC. After that, we use a greedy feature selection algorithm to generate the best IoC for this malware.
– We introduce the method to generate structured CTI following STIX standard, using information from the unstructured malware traces.

## 2 Background

In this section, we will first introduce the format of IoC and then introduce the OpenIoC standard using in our IoC generation process. We then explain terms of CTI and the Structured Threat Information Expression (STIX).

### 2.1 Indicators of Compromise

Indicators of compromise is a set of proofs that can be used to identify intrusion on a host or network [19], and now has become one of the most potent threat events detection weapons [26]. An IoC expression consists of *IndicatorItem*, two kinds of condition "`is`" and "`contains`" and a corresponding value field. IoC use operators "`AND`" or "`OR`" to combine several IoC expressions, making it possible to describe the abnormal traces in multiple dimensions. We can identify an intrusion if the traces or logs obtained from the network or host match the IoC condition. OpenIoC is a simple XML based IoC describing schema advised by MANDIANT. It is convenient for us to generate structured IoC following OpenIoC standard, as it only requires six necessary XML tags and simple express grammar. Also can we transform OpenIoC expression into other IoC describing standard.

Besides using hundreds of pre-defined *IndicatorItem* to describe behavior or threat in detail, OpenIoC allows users to define their *IndicatorItem*. This also convenience our IoC generation process, as not all the features can be classified into suitable pre-defined *IndicatorItem* in the analysis of malware traces.

### 2.2 Cyber Threat Intelligence and STIX

Cyber Threat Intelligence (CTI) is defined as "evidence-based knowledge, including context, mechanisms, indicators, implications, and actionable advice, about an existing or emerging menace or hazard to assets that can be used to inform decisions regarding the subject's response to that menace or hazard", according to Gartner [20].

To convenience the sharing of CTI and let CTI can be understood by the machine, there are many CTI describe languages like STIX [24], CybOX [4], and MAEC [5] is created. STIX is one of the most expressive CTI describe standards created by MITRE. Users can describe different elements in a cyber threat, including basic network features, IoC, TTP and exploited vulnerabilities. All these messages can be organized in a `json` file. STIX2.0 defines 12 types of

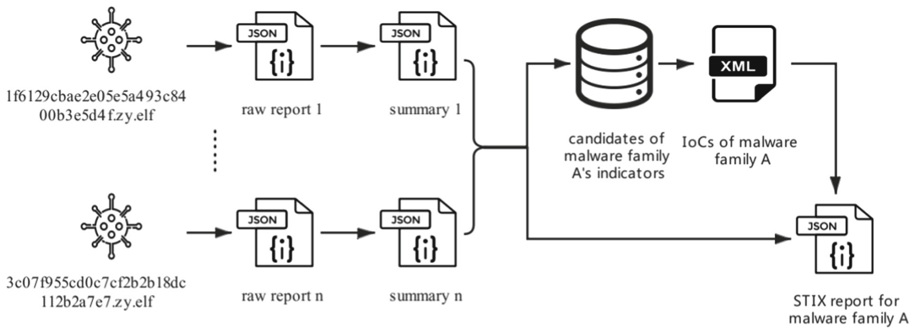


**Fig. 1.** Overview of generating CTI from malware samples obtained from the Internet

STIX Domain Objects (SDOs) to describe different types of threat intelligence, and 2 STIX Relationship Objects (SROs) to describe the relationship between different SDOs. STIX is more expressive than OpenIoC, as STIX can involve more information except threat indicators.

## 3 Overview

In this part we will give an overview of how we generate CTI following STIX from malware obtained from the Internet. Figure 1 illustrates how our system works. For one kind of malware class, we are first using sandbox analysis to obtain the malware traces. We then perform our IoC generation algorithm after malware traces processing, in which we select several kinds of strings and transform them into IoC expressions. Finally, we use generated IoC expressions and other useful information in the summary of traces to organize a STIX report. Works done in each step will be described below.

**Malware Analysis.** We use a sandbox to finish our malware analysis mission. It is worth noting that the structure and content in the sandbox reports will not influence our system and results. Though a sandbox can not simulate all the running conditions of malware samples, the information we can get from sandbox reports is enough to extract malware features. For every malware sample that belongs to the same class, we get its running traces through sandbox simulation. The sandbox analysis result will be written in a `json` file.

**Data Preprocessing.** There are two kinds of information in the malware traces: static analysis results and dynamic analysis results. The static analysis results include basic file information like file name, file MD5, YARA matching results, and strings obtained from reverse analysis; The dynamic analysis results are logs of interaction between malware sample and host or network, for example, network traces, process creation, file creation and delete behavior and register modify.

Not all the strings extracted from malware are useful for IoC generation. Figure 2 shows part of the sandbox analysis result which are organized in a `json` file. The `string` block gives the result of static string extraction, in which we observe that only a very small part of these string extraction outputs like command or DLL name is meaningful and useful. Most of these extracted strings are hard to understand and meaningless; therefore, we choose to discard these meaningless or repeated strings in malware traces preprocessing.

```
"static_info": {
  "scan": {...},
  "strings": [
      "+UntAsyncTask",
      "SetWindowPlacement",
      "pmCopy",
      "TIntConst",
      "VarMul",
      "WaitForSingleObject",
      "TSampleGrabberCBInt",
      "RUSSIAN_CHARSET",
      "OnContextPopup",
      ...]
}
"behavior": {
  "basic_behavior": {...},
  "behavior_sequence": [...],
  "other_behavior": [...]
}
```

**Fig. 2.** Part of sandbox analysis result

On the other hand, not all the analysis results have the same value. When we try to generate concise and robust IoC and CTI, we should discard those low-value strings in malware traces. File MD5 is undoubtedly efficient and accurate in malware detection, but file MD5 based malware detection needs many active samples and has a short life circle. If we use MD5 as our indicator, the attacker can bypass this IoC detection at ease [6]. Although most open source IoC use file MD5 as their indicator, we will not use it in our work. Similar low-value information also includes file compilation time and file size.

**IoC Generation.** After data preprocessing, we obtain a shorter version of malware traces, leaving only strings that can better characterize the behaviors or features of malware in the sandbox simulation. In order to generate structured IoC, we first transform strings in sandbox analysis result into IoC expression. For a single piece of malware behavior description or static characteristic information, we select appropriate *IndicatorItem* according to the content and construct a IoC expression. We will talk about the transformation rules in Sect. 4.1.

We then count all the IoC expressions in each malware trace to build a candidate list for each malware class. For those IoC expressions which have similar value field, we generate a regular expression to replace them. We will describe why and how we construct the candidate list in Sect. 4.2. In the last phase of IoC generation, we choose the most suitable IoC expression subset from the candidate list through a feature selection algorithm. Our target here is to select a subset that has the best performance on the dataset. We will introduce our algorithm and metrics in Sect. 4.3.

**STIX Report Generation.** Besides the information we used to generate IoC, there are also other strings like network traffic and basic observation in the malware analysis report, which can help understand and characterize certain type of malware. Furthermore, we can summarize malicious behavior set from malware traces, which can help the malware detection and build a better defense system. We organize analysis results of a malware class following STIX standard. We select several SDOs suitable for our task, including *Indicator*, *Observed Data*, *Malware*, *Threat Actor*, *Identify*, *Intrusion Set* and *Report*. We design several SROs to describe the relationship between different SDOs and provide a visualization version of the STIX report. Details about how we construct SDOs and design the SROs will be shown in Sect. 5.

## 4 IoC Generation

In this section, we will describe how we generate IoC from malware traces in detail.

### 4.1 Expression Transform

Transforming strings in malware traces into a standardized form will benefit the candidate list building process. To transform static and dynamic features in malware traces into IoC expression, we first build the conversion relationship between feature strings and *IndicatorItem*. OpenIoC provides 27 pre-defined *IndicatorItem*, each of them can be divided into more detailed indicators according to specific behavior or feature. Table 1 gives several examples of expression-*IndicatorItem* conversion.

For every string in malware traces, search the corresponding *IndicatorItem* and use its content to fill the value field of IoC expression. There are also some descriptions in malware traces that cannot be expressed with only one simple *IndicatorItem*, as we showed in Table 1. In this case, we find out all the corresponding *IndicatorItem* and use **AND** to connect these IoC expressions.

The predicate(*is*, *contains* or *matches*) should be used in the IoC expression can be determined once we choose a specific *IndicatorItem*. As a result, it is not necessary to use the predicate *not* in our IoC expression.

### 4.2 Candidate List Generation

We design a candidate list for certain malware classes to record the IoC expressions and their occurrence frequency. Every IoC expression converted from the

**Table 1.** Examples of expression conversion

| Description in malware traces | OpenIoC *IndicatorItem* |
|---|---|
| MD5 | FileItem/File MD5 |
| filename | FileItem/File Filename |
| file_behavior add | FileItem/File Filename Created |
| reg_behavior new | RegistryItem/Registry Key Path **AND** RegistryItem/Registry Value **AND** RegistryItem/Registry Value Name |
| process created | ProcessItem/Process PID **AND** ProcessItem/Process |

malware traces will be stored in a `.csv` file in the form of (logical predicate, indicator type, indicator subdivision type, expression value type, expression value, frequency) six-tuple form. The constructed candidate list can facilitate the classification and matching of IoC expressions. Besides, malware may produce variants shortly, with means generated IoC may not maintain high efficiency. With the help of IoC expressions candidate list, we can easily add newly obtained data in after initialization. This design guarantees that our generated IoC can indicate latest malware sample and its attack behavior.

It is challenging to detect and handle those highly similar IoC expressions when building a candidate list. Since malware samples may have different versions, not all of their behaviors are entirely consistent. For instance, malware in a specific class may all write data into other processes, but the target process may not exactly be the same. This will result in two different IoC expressions after our data preprocessing and expression transform, however, the two IoC expressions are representing the same kind of behavior or malware characteristic, with only very little difference in the target process name. Recent studies have proposed several methods to solve this problem using cluster algorithm [28] or multi-granular regular expression extraction [11]. We find that these methods, especially clustering based methods, may not distinguish different kinds of behaviors, and eventually result in bad performance.

Similar IoC expressions seldom appear in our obtained malware traces. To address this challenge, we divide IoC expressions in the candidate list according to their types(for example, file behavior, register edit and so on). In every divided subset, select the expressions with high frequency as the aggregation center, then calculate the *Levenshtein Distance* between high-frequency expressions and other expressions in the same subset. If the calculated distance between two expressions smaller than our prescribed threshold, use a regular expression to merge them. We use a heuristic method to decide the threshold for each class of malware, which mainly depends on the average length of IoC expressions' value field.

Figure 3 shows an example of how we use regular expression to merge similar IoC expressions. As we mentioned before, we will only calculate the Levenshtein

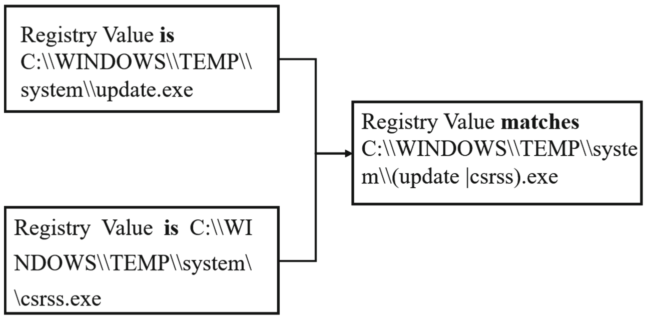

Registry Value **is** C:\\WINDOWS\\TEMP\\system\\update.exe
Registry Value **is** C:\\WINDOWS\\TEMP\\system\\csrss.exe
Registry Value **matches** C:\\WINDOWS\\TEMP\\system\\(update |csrss).exe

**Fig. 3.** A sample of using regular expression to merge similar IoC expressions

Distance for those IoC expressions with high frequency to reduce the computational cost. When we use a regular expression to replace the value field in IoC expression, we use *matches* as the predicate. To control the complexity of IoC expressions, we only use — to construct a regular expression.

## 4.3 IoC Expression Selection

Effective IoC should have properties that only identify attacker's activities, easy to evaluate and expensive for attackers to evade, according to MANDIANT [19]. In other words, the IoC we expected should have a high true positive rate and low false positive rate, and the indicators and value field we designed should be concise and easy for a machine to understand.

IoC subset selection is similar to the feature selection problem. Suppose there are $\mathtt{N}$ candidate IoC expressions in the candidate list; our goal is to select a subset which has the best performance and the highest quality on our malware traces dataset. In this problem, the search space will be $2^{\mathtt{N}} - 1$. We can easily evaluate one specific IoC subset, but we may need to test all combinations to find the best one. By using a suitable algorithm, we can obtain an approximate optimal solution of this problem. For example, [28] builds a keyword tree base on the frequency of IoC attributes and find out IoC from the root node to specific leave nodes; [11] designs a submodular function to evaluate the performance of IoC, and generates IoC through maximizing the submodular function [12].

Compared with feature selection problem in machine learning tasks, we have a priori knowledge about the frequency of candidate IoC expressions, which can represent the importance of this IoC expression, in other words, the higher frequency an IoC expression has, the better the IoC including this expression perform precision and recall. We design a heuristic IoC selection algorithm *GIG* which starts the selection from the IoC expression with the highest frequency:

Function $p()$ in the algorithm will evaluate the precision performance of current IoC subset on the test set, $n$ represents the current scale of generated IoC subset $\mathcal{C}$. Our algorithm first adds one of the IoC expressions with the highest frequency into IoC subset, then continuously adds expressions that can improve

**Algorithm 1.** Greedy IoC Generation

```
Input: Dataset $\mathcal{D}$, Candidate list $\mathcal{I}$, maximum expression number m, precision threshold $\theta$
Output: IoC subset $\mathcal{C}$
function EXPRESSION SELECTION($\mathcal{D}, \mathcal{I}, m, \theta$)
    add the most frequent $i \in \mathcal{I}$ to $\mathcal{C}$
    $n \leftarrow 1$
    $\mathcal{I} \leftarrow \mathcal{I} - i$
    while ($p(\mathcal{C}) <= \theta \quad || \quad n < m$) do
        find i from $\mathcal{I}$ until $p(\mathcal{C} \cup i) > p(\mathcal{C})$
        if successfully find $i$ then
            $\mathcal{C} \leftarrow \mathcal{C} \cup i$
            $\mathcal{I} \leftarrow \mathcal{I} - i$
            $n \leftarrow n + 1$
        end if
        if can not find any $i$ then
            return $\mathcal{C}$
        end if
    end while
    return $\mathcal{C}$
end function
```

IoC subset's performance, until the IoC expressions in IoC subset hit the prescribed maximum expression number $m$ or the IoC subset is good enough. When we find more than one IoC expression which brings the same precision increase in one iteration, we randomly add one of them into our IoC. We set the maximum expression number $m$ to control the conciseness of the final IoC. Our experiment shows that we can get concise and effective IoC through fixing a suitable $m$.

With the help of *IoC writer* [18], we can easily export the IoC from tabular data to an OpenIoC format `.IoC` file. The selected expressions $(i_1, i_2, ...i_n)$ will be connected with `OR`, and the final IoC for a specific kind of malware will be like $(i_1||i_2||...||i_n)$. Figure 4 shows the IoC our algorithm generated for DarkComet family.

(Service API **matches** (SeDebugPrivilige—SeLoadDriverPrivilige))
**OR** (Hook Hooked Module **matches** C:\\WINDOWS\\TEMP\\(csrss—msdcsc).exe)
**OR** (Registry Key Path **contains** HKEY_LOCAL_MACHINE\\Software\\Microsoft \\Windows NT\\CurrentVersion\\Winlogon
**AND** Registry Key Value name **contains** UserInit
**AND** Registry Key Value **matches** C:\\WINDOWS\\TEMP\\system\\(update—csrss —msdcsc).exe)

**Fig. 4.** Generated IoC for DarkComet following OpenIoC standard

## 5 STIX Report Construction

When constructing a STIX report, we first fill each SDOs, and then define SROs between SDOs to describe their relationship. This section will show the details of how we select SDOs and SROs for STIX report generation task and discuss how to use information in malware traces to construct SDOs and SROs.

### 5.1 SDOs Selection and Construction

Document [24] defines the description scope and format of all 12 types of SDOs. As we mentioned before, to organize these scattered and unstructured malware traces, we need to select SDOs suitable for our description scenario. Specifically, we choose the following SDOs to construct STIX report:

- *Observed Data:* This SDO contains a series of monitored basic behavior or information like file name, IP address and network traces. For specific malware class, we directly write corresponding basic observation in all sandbox reports into an Observed Data object after merging similar items.
- *Malware:* Malware object describes malware category and its common characteristics. Though the Malware object itself can only describe some general characteristics, we can link it to other SDOs we use to depict a vast landscape of malware attacks.
- *Threat Actor:* This SDO can describe the information about individuals or organizations related to threat events. We treat each malicious IP address as an attacker in our malware description task and write an IP address list into threat actor object. Malware likes bots [27] may try to connect back to the control side during the sandbox simulation, we can record those malicious IP addresses by analyzing the network traces. There is another SDO in STIX, Identity, which can describe the information of attackers. However, the Identify object should contain information of victims at the same time, as a result, Threat Actor is the better choice for our description task.
- *Indicator:* Indicator is an essential component in the malware CTI, consistent and well structured IoC can help to automate some processes in malware detection. STIX provides more expressive IoC describe grammar, compare to OpenIoC. Using the API developed by OASIS [23], we can transform our generated IoC into STIX's Indicator format.
- *Intrusion Set:* Intrusion Set summarizes the malicious behavior of malware during the sandbox simulation, such as process hijacking, registry modification. We infer malicious behavior from the malware traces and use these results to construct an Intrusion Set object in our CTI report.
- *Report:* The Report object in STIX organizes all the related SDOs together. A list of references and descriptions to other SDOs will be written in.

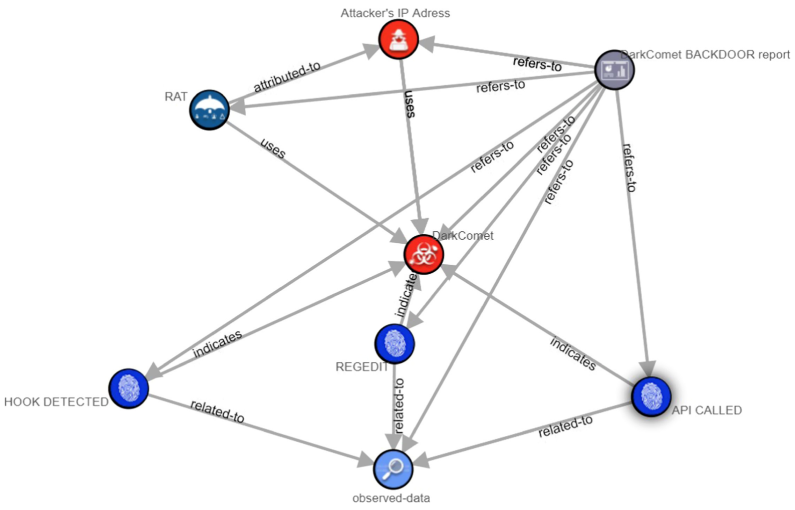


**Fig. 5.** Visualized STIX report of DarkComet

## 5.2 SROs Construction

In our malware CTI generation process, we first obtain unstructured malware traces from sandbox analysis, then we finish work like malware signature and indicator construction, attacker identify and malicious behavior analysis. Our SROs established based on this scenario are as follows:

- Observed Data **related-to** Indicator;
- Indicator **indicates** malware;
- Threat Actor **uses** malware;
- Intrusion Set **uses** malware;
- Intrusion Set **attributed-to** Threat Actor.

The **related-to**, **indicates**, **uses** are connection types defined by STIX [24]. Finally we get a malware CTI heterogeneous information network consists of SDOs and SROs with the help of a STIX visualization toolkit [22], as showed in Fig. 5.

## 6 Evaluation

In this section, we will introduce a series of experiments to evaluate whether our IoC generation algorithm can produce concise and effective IoC for specific malware class. We will first introduce our dataset and settings. After that we will introduce our evaluation methodology and give our experiment results.

## 6.1 Evaluation Setup

We deploy a honeypot system to collect malware samples from the Internet from May 2019 to June 2020. We use ANTIY's [1] sandbox service to get malware traces. We build a Windows XP 32 bit and a centos 6.5 32 bit virtual machine as the sandbox analysis environment. The structure and content in the sandbox reports has been described in Sect. 3 and Sect. 4. Table 2 summarizes the malware samples used in our experiment, and shows the analysis environment we use. We label these malware samples with the help of YARA rules.

**Table 2.** Malware samples used in IoC generation algorithm evaluation

| Malware | Size | Environment |
|---|---|---|
| njRat | 259 | win XP 32 bit |
| DarkComet | 113 | win XP 32 bit |
| NanoCore | 174 | win XP 32 bit |
| Setag | 172 | centos 32 bit |
| Gafgyt | 252 | centos 32 bit |

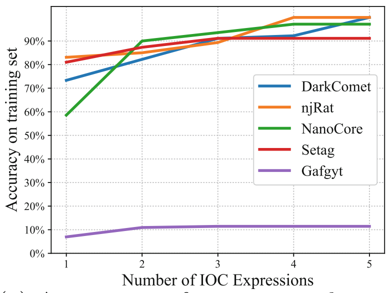
To evaluate the detection efficiency of our generated IoCs, we split these malware samples into two parts to simulate a real-world malware detection scene. The first part, namely the training set, consists of 80% of malware samples; we will use them to generate IoC for each malware class. We use the rest of malware samples as testing objects.

The analyzer machine runs only one malware sample at the same time to avoid interference, and the running report will be written in a `json` file. We build a candidate list for each malware class with our method using samples in the training set. We then perform *GIG* to generate one IoC for each malware class; Fig. 4 shows one generation result. We run all the experiments on 8 cores Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.20 GHz with 16 GB RAM.

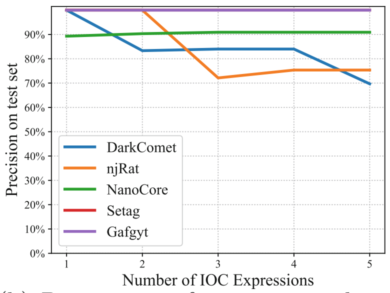
In the first part of the evaluation, we will investigate whether IoCs generated from the training set can detect unknown testing objects. After data preprocessing, we judge whether a testing object belongs to specific malware classes using generated IoCs. We also evaluate whether our IoCs can achieve the same classification efficiency as other methods, especially machine learning based methods.

## 6.2 IoC Generation Evaluation

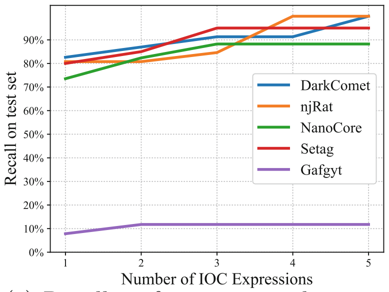
In this experiment, we first use the training set to generate IoC for each malware class, then we evaluate those IoC on the test set consisting of all five types malware samples. The evaluation of IoC can be considered a binary classification problem, and we should be concerned about the precision and recall rate of generated IoC. An IoC with high precision and recall rates means lower false positive and false negative in real world malware detection, which better meets our needs.



(a) Accuracy performance on the training set



(b) Precision performance on the test set



(c) Recall performance on the test set

**Fig. 6.** The performance of generated IoC w.r.t. number of IoC expressions

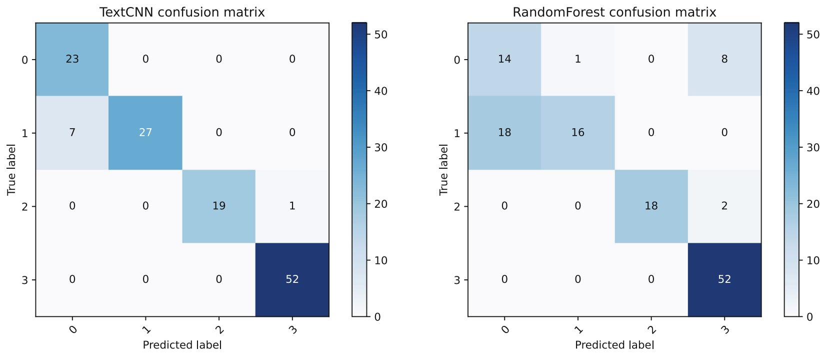
Figure 6 illustrates the performance of generated IoC w.r.t. number of IoC expressions on different dataset. Not all the malware samples will have dynamic behavior during the sandbox simulation due to the imperfection of the sandbox running mechanism. For traces obtained from these inactive malware samples, since we choose to discard most of the static characteristics in the traces preprocessing, there will be very few strings in it and lack of common feature, which may lead to bad algorithm performance. For example, we find that nearly all the Gafgyt samples in our experiment are inactive during the sandbox simulation; as a result we get a lousy curve for it.

For other malware classes, as we generate our IoC respectively, the accuracy performance on the training set is the same as precision performance. Our algorithm continuously adds expression into IoC, we can see in Fig. 6(a) that the more expressions we use, the higher accuracy we achieve. As we mentioned in Sect. 4.1, we generate candidate IoC expressions from every malware trace. To evaluate generated IoC on test set, we compare it with every sample's IoC expressions to see whether this IoC can detect target malware sample. Figure 6(b) and 6(c) show the performance of generated IoC on test set. Similar to the accuracy performance on training set, the more expressions we use, the higher recall rate we get. However, more expressions may not bring higher precision performance, since the more expressions we use, the more false positive will appear. According to our experiment result, if we add no more than 3 expressions into the IoC, we can get concise and effective IoC. Table 3 shows the performance of 3 item IoC on our test set. In conclusion, our system is practical enough to generate effective malware threat intelligence and help malware detection.

**Table 3.** Recall performance on test set of 3 items IoC

| Malware | Recall |
|---|---|
| njRat | 91.31% |
| DarkComet | 84.62% |
| NanoCore | 88.24% |
| Setag | 95.00% |

## 6.3 Algorithm Comparison



(a) TextCNN confusion matrix (b) RandomForest confusion matrix

TextCNN confusion matrix:

|  | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 23 | 0 | 0 | 0 |
| 1 | 7 | 27 | 0 | 0 |
| 2 | 0 | 0 | 19 | 1 |
| 3 | 0 | 0 | 0 | 52 |

RandomForest confusion matrix:

|  | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 14 | 1 | 0 | 8 |
| 1 | 18 | 16 | 0 | 0 |
| 2 | 0 | 0 | 18 | 2 |
| 3 | 0 | 0 | 0 | 52 |

**Fig. 7.** Classification performance of machine learning based algorithms

Detection accuracy is one of the critical metrics for IoC. Generally, machine learning based malware classification model can utilize implicit features to achieve better classification performance, comparing to coarse-grained method likes expression and regular expression based model. In this comparison, we will investigate whether IoC generated by *GIG* can achieve the same classification performance as machine learning methods, with the help of data preprocessing and fine-grained feature selection.

To compare our algorithm's classification performance with machine learning based method, we set up Random-Forst [2] and TextCNN [10] models on our dataset as the opponents. The TextCNN used in our experiment has four kernels of different sizes (2, 3, 4 and 5) and has 800 kernel in total. In the embedding layer, we first train a word2vec model [21] to generate word vector from malware traces and change every trace into a matrix. On the other hand, we treat each expression in candidate list as a feature to build feature vector and train a RandomForest classifier with 100 estimators. Figure 7 shows the confusion matrix of RandomForest and TextCNN models in the classification task.

Table 4 shows the recall rate comparison between different algorithms; we run our algorithm to generate IoC with three expressions in the comparison. The results indicate our method performs at least not worse than general machine learning based algorithms. Simultaneously, our method has the obvious advantage of generating reliable, concise and easy for machine to understand IoC which can help the malware detection in the real environment. On the contrary, those machine learning based algorithms can only work like a black box classifier, which the classification criteria behind is hard to explain.

**Table 4.** Recall rate comparison

| Method | Malware | | | |
|---|---|---|---|---|
| | DarkComet | NanoCore | Setag | njRat |
| GIG | 91.31% | 88.24% | 95% | 84.62% |
| TextCNN | 100% | 79.41% | 95% | 100% |
| RandomForest | 60.87% | 47.06% | 90% | 100% |

## 7 Related Work

**Cyber Threat Intelligence.** Traditional CTI generation methods rely on security experts to summarize the large amount but low-value basic network traces or system logs, which is inefficient and labor-consuming. More and more research is focusing on automatically generating CTI, especially IoC extraction using different methods. Published works like [15] explore how IoC related information is described in security articles and reports, and develop a NLP model to automatically extract IoC from them; [31] designs a multi-granular attention based IoC recognition system based on BiLSTM and CRF to extract IoC entities from security blogs and articles. Many works try to extract IoC from Open Source Intelligence(OSINT), though their data source may be different and may develop different generation models based on machine learning and regular expression [3]. These OSINTs are somehow well structured and IoC information will be described in a particular format; as a result, it is possible to extract them through NLP algorithms. If we want to obtain IoC as timely as possible, we may need to extract them from unstructured malware or threat event traces, and we need further study to automate this process.

Besides IoC, there is still lots of information in malware or network traces that can help depict a complete picture of threat events. Related work like [9] try to automatically construct attack pattern messages from OSINT and organize them into CTI expression standards. In our work, we extract threat information about attackers and intrusion sets from malware traces and organize them following STIX standard.

Another point in CTI research is to evaluate the actual effect of those automatically generated IoC or CTI and utilize them in real world threat action detection. [14] designs a set of evaluation metrics and measures a broad range of Threat Intelligence(TI) and concludes that there are still many challenges and limitations in using existing IoC and CTI.

**Malware Detection.** Pioneer work has made remarkable improvements in malware detection. The malware detection process can be divided into feature selection and classification/clustering [30]. In the feature selection stage, fine-grained features are extracted to improve detection performance. For example, [17] summarizes three characteristics in bots' network traces and designs signature for detection; [25] focuses on HTTP-based malware and defines similarity metrics to build a network level malware clustering system. In the classification and clustering stage, researchers introduce different models in recent years. For example, [8,16] build a heterogeneous information network based on host logs, and use graph embedding algorithm to construct eigenvector for every node, then find out malicious log entries through clustering; [29] first derives the CFGs of malware file and use graph based algorithm to finish malware classification.

We use sandbox analysis reports as our source data to make up for the lack of static characteristics in malware detection and IoC generation [7]. We generate IoC and CTI from malware traces analysis, of course the generated IoC can be used to finish the malware classification task.

## 8 Conclusion

This paper introduces a system to automatically generate IoC and CTI following STIX standard from unstructured malware traces. Different from OSINT based CTI or IoC extraction research, we design our system to generate them directly from malware samples' sandbox analysis reports. In the IoC generation phase, we first define rules for the transformation between strings in malware traces and IoC expression. After that, for a specific class of malware, we build an IoC expressions candidate list. Finally we propose *GIG* to select the most effective IoC from the candidate list. Our experiment of IoC generation achieves a 89.4% recall rate on average. In conclusion, our system is practical enough to generate concise and effective malware IoC and threat intelligence, which can help the real world malware detection and security operation.

**Acknowledgement.** We thank the anonymous reviewers for the valuable comments and suggestions. This work is supported by National Key Research and Development Program of China [No. 2020YFB1807500] and National Natural Science Foundation of China [No. 61831007].

## References

1. ANTIY. https://www.antiy.cn/
2. Breiman, L.: Random forests. Mach. Learn. **45**(1), 5–32 (2001)
3. Catakoglu, O., Balduzzi, M., Balzarotti, D.: Automatic extraction of indicators of compromise for web applications. In: Proceedings of the 25th International Conference on World Wide Web, WWW 2016, International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, pp. 333–343 (2016). https://doi.org/10.1145/2872427.2883056
4. Corporation, T.M.: CybOX: cyber observable expression. https://cyboxproject.github.io
5. Corporation, T.M.: Malware attribute enumeration and characterization (MAEC). https://maecproject.github.io/documentation/overview/
6. David, B.: The pyramid of pain: Intel-driven detection & response to increase your adversary's cost of operation. Technical Report, FireEye. https://rvasec.com/slides/2014/Bianco_Pyramid
7. Firdausi, I., Erwin, A., Nugroho, A.S., et al.: Analysis of machine learning techniques used in behavior-based malware detection. In: 2010 Second International Conference on Advances in Computing, Control, and Telecommunication Technologies, pp. 201–203. IEEE (2010)
8. Gao, Y., Li, X., Peng, H., Fang, B., Yu, P.: HinCTI: a cyber threat intelligence modeling and identification system based on heterogeneous information network. IEEE Trans. Knowl. Data Eng., 1 (2020)
9. Husari, G., Al-Shaer, E., Ahmed, M., Chu, B., Niu, X.: TTPDrill: automatic and accurate extraction of threat actions from unstructured text of CTI sources. In: Proceedings of the 33rd Annual Computer Security Applications Conference, pp. 103–115 (2017)
10. Kim, Y.: Convolutional neural networks for sentence classification. arXiv preprint arXiv:1408.5882 (2014)
11. Kurogome, Y., et al.: EIGER: automated IOC generation for accurate and interpretable endpoint malware detection. In: Proceedings of the 35th Annual Computer Security Applications Conference. ACSAC 2019, pp. 687–701. Association for Computing Machinery, New York (2019). https://doi.org/10.1145/3359789.3359808
12. Lakkaraju, H., Bach, S.H., Leskovec, J.: Interpretable decision sets: a joint framework for description and prediction. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1675–1684 (2016)
13. Li, J.: Cyberspace threat intelligence perception, sharing and analysis technology: A survey. Chin. J. Netw. Inf. Secur. **2**(002), 16–29 (2016)
14. Li, V.G., Dunn, M., Pearce, P., McCoy, D., Voelker, G.M., Savage, S.: Reading the tea leaves: a comparative analysis of threat intelligence. In: 28th {USENIX} Security Symposium ({USENIX} Security 19), pp. 851–867 (2019)
15. Liao, X., Yuan, K., Wang, X., Li, Z., Xing, L., Beyah, R.: 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, pp. 755–766 (2016)
16. Liu, F., Wen, Y., Zhang, D., Jiang, X., Xing, X., Meng, D.: Log2vec: a heterogeneous graph embedding based approach for detecting cyber threats within enterprise. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, pp. 1777–1794 (2019)

17. Liu, L., Chen, S., Yan, G., Zhang, Z.: BotTracer: execution-based bot-like malware detection. In: Wu, T.-C., Lei, C.-L., Rijmen, V., Lee, D.-T. (eds.) ISC 2008. LNCS, vol. 5222, pp. 97–113. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85886-7_7
18. MANDIANT: IOC writer. https://github.com/mandiant/ioc_writer
19. MANDIANT: Sophisticated indicators for the modern threat intelligence: an introduction to openIOC. Technical Report, MANDIANT. https://www.academia.edu/31820654/An_Introduction_to_Open_IOC
20. McMillan, R., Pratap, K.: Market guide for security threat intelligence services. Technical Report, Gartner (2014). https://www.gartner.com/en/documents/2874317
21. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781 (2013)
22. OASIS: Lightweight visualization for STIX 2.0 objects and relationships. https://github.com/oasis-open/cti-stix-visualization/
23. OASIS: openIOC-to-SITX. https://github.com/STIXProject/openioc-to-stix
24. OASIS: STIX version 2.0. part 2: STIX objects. Technical Report. https://docs.oasis-open.org/cti/stix/v2.0/stix-v2.0-part2-stix-objects.pdf
25. Perdisci, R., Lee, W., Feamster, N.: Behavioral clustering of http-based malware and signature generation using malicious network traces. In: NSDI, vol. 10, p. 14 (2010)
26. SANS: The sans state of cyber threat intelligence survey: CTI important and maturing. Technical Report. https://www.sans.org/reading-room/whitepapers/analyst/state-cyber-threat-intelligence-survey-cti-important-maturing-37177
27. Stinson, E., Mitchell, J.C.: Characterizing bots' remote control behavior. In: M. Hämmerli, B., Sommer, R. (eds.) DIMVA 2007. LNCS, vol. 4579, pp. 89–108. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-73614-1_6
28. Xu, W., Wang, Y., Xue, Z.: Automatic generation of IOC for threat intelligence. Commun. Technol. **50**(1), 116–123 (2017)
29. Yan, J., Yan, G., Jin, D.: Classifying malware represented as control flow graphs using deep graph convolutional neural network. In: 2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pp. 52–63. IEEE (2019)
30. Ye, Y., Li, T., Adjeroh, D., Iyengar, S.S.: A survey on malware detection using data mining techniques. ACM Comput. Surv. (CSUR) **50**(3), 1–40 (2017)
31. Zhao, J., Yan, Q., Liu, X., Li, B., Zuo, G.: Cyber threat intelligence modeling based on heterogeneous graph convolutional network. In: 23rd International Symposium on Research in Attacks, Intrusions and Defenses ({RAID} 2020), pp. 241–256 (2020)