EIGER: Automated IOC Generation for Accurate and Interpretable Endpoint Malware Detection

Yuma Kurogome
NTT Secure Platform Laboratories
yuuma.kurogome.gk@hco.ntt.co.jp

Yuto Otsuki
NTT Secure Platform Laboratories
yuuto.ootsuki.uh@hco.ntt.co.jp

Yuhei Kawakoya
NTT Secure Platform Laboratories
yuuhei.kawakoya.sy@hco.ntt.co.jp

Makoto Iwamura
NTT Secure Platform Laboratories
makoto.iwamura.sw@hco.ntt.co.jp

Syogo Hayashi
NTT Security (Japan) KK
Syogo.Hayashi@nttsecurity.com

Tatsuya Mori
Waseda University / NICT
mori@nsl.cs.waseda.ac.jp

Koushik Sen
University of California, Berkeley
ksen@berkeley.edu

ABSTRACT

A malware signature including behavioral artifacts, namely Indicator of Compromise (IOC) plays an important role in security operations, such as endpoint detection and incident response. While building IOC enables us to detect malware efficiently and perform the incident analysis in a timely manner, it has not been fully-automated yet. To address this issue, there are two lines of promising approaches: regular expression-based signature generation and machine learning. However, each approach has a limitation in accuracy or interpretability, respectively.

In this paper, we propose *EIGER*, a method to generate interpretable, and yet accurate IOCs from given malware traces. The key idea of EIGER is *enumerate-then-optimize*. That is, we *enumerate* representations of potential artifacts as candidates of IOCs. Then, we *optimize* the combination of these candidates to maximize the two essential properties, *i.e.,* accuracy and interpretability, towards the generation of reliable IOCs.

Through the experiment using 162K of malware samples collected over the five months, we evaluated the accuracy of EIGER-generated IOCs. We achieved a high True Positive Rate (TPR) of 91.98% and a very low False Positive Rate (FPR) of 0.97%. Interestingly, EIGER achieved FPR of less than 1% even when we use completely different dataset. Furthermore, we evaluated the interpretability of the IOCs generated by EIGER through a user study, in which we recruited 15 of professional security analysts working at a security operation center. The results allow us to conclude that our IOCs are as interpretable as manually-generated ones. These results demonstrate that EIGER is practical and deployable to the real-world security operations.

CCS CONCEPTS

• **Security and privacy → Malware and its mitigation**.

KEYWORDS

Malware, Detection, Classification, Indicator of Compromise

ACM Reference Format:
Yuma Kurogome, Yuto Otsuki, Yuhei Kawakoya, Makoto Iwamura, Syogo Hayashi, Tatsuya Mori, and Koushik Sen. 2019. EIGER: Automated IOC Generation for Accurate and Interpretable Endpoint Malware Detection. In *2019 Annual Computer Security Applications Conference (ACSAC '19), December 9–13, 2019, San Juan, PR, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3359789.3359808

ACSAC '19, December 9–13, 2019, San Juan, PR, USA

ACM ISBN 978-1-4503-7628-0/19/12…$15.00
https://doi.org/10.1145/3359789.3359808

1 INTRODUCTION

Indicator of Compromise (IOC) [11, 12, 19] is a malware signature representing "artifacts" generated along with its execution. That includes file paths, registry keys, process handles, command lines, IP address, domain name, and *etc.* IOCs are widely used in the real-world security operations, such as Security Operation Center (SOC) or Computer Security Incident Response Team (CSIRT), to automate a part of analysts' tasks for endpoint detection and incident response. The use of IOCs *i.e.,* scanning endpoints with IOCs and validating false positives is now incorporated into their daily routine. Since IOCs are mainly used in such a human-in-the-loop situation, the interpretability of them becomes an important factor for analysts, as well as the accuracy of detection.

Even though the importance of IOCs becomes larger day by day, the procedure of generating IOCs has not been well-developed. In fact, we have to heavily depend on analysts' experiences and intuitions to find out the artifacts that can clearly distinguish specific malware from the others. In other words, the procedure entails manual work and is not fully-automated. As a result, we give attackers a huge advantage over a cat and mouse race. Thus, we are forced to fall into the situations that we do not have enough IOCs for protecting our premises from attackers.

To automate the procedure, there are two promising approaches: regular expression-based and machine learning-based ones. In the former, we can borrow the idea of generating a signature from malware traffic for network-level detection [40, 44, 55], while an endpoint is their out of scope. In their manner, we first extract common strings appeared in multiple malware traffic, and then convert each of the similar part in a common string into a regular expression to approximate similar malware traffic. However, when we naively

apply this approach for endpoint artifacts, we may generate too-abstract regular expression patterns. For instance, lets take a look at a scenario in which we want to generate an IOC for "aareg.exe" and "install.vbs" under the temporal directory of a Windows operating system. Widely adopted expression-based approach such as [44] may generate "C\:\\Users\\.*\\AppData\\Local\\Temp\\.*". Clearly, this IOC will cause false positives because it is too generic. An IOC we expect to generate here is like "C\:\\Users\\.*\\AppData\\Local\\Temp\\(aareg\.exe|install\.vbs)", which is triggered by only specific files under the temporal directory to suppress false positives. In summary, while regular expression itself is promising for the IOC generation, the existing expression-based methods as they are cannot be diverted to our method. This is because they generate regular expressions only at a single level of abstraction.

In the latter, we allocate weights to strings contained in either malware binaries or traces, and then train a model that represents malware as a (non-)linear transformation of those weights. For instance, Decision Tree [5] trained over the aforementioned example would be as follows: if get_weight("AppData") > 0.37 and if get_weight("Local") > 0.48 and if not get_weight("Local") > 0.6 and if get_weight("Temp") > 0.53 ... Such an approach could generate a model for detecting artifacts with high accuracy, but this may generate a too complex IOCs for human-analysts to understand, even if the malware can be detected by the simple condition. In addition, this kind of approach trains a single model on the entire dataset given, thus it mixes decision boundaries for each family. This complexity may result in a time loss of triage and analysis since these process include human-interactions, as we explained above. In summary, these two approaches are promising as an automatic IOC generation method, but should not be directly applied because they generate excessively abstracted and low interpretable IOCs, respectively. These limitations motivate us to propose a new method to tackle the challenge, *i.e.,* generating interpretable IOCs with satisfying the same detection accuracy as machine learning approaches that pursue only the detection accuracy.

In this paper, we propose Exhaustive IOC Generator, *EIGER* for short, a method to automatically generate accurate and interpretable IOCs from given malware analysis results. EIGER consists of two key steps: *Candidate Enumeration* and *Optimization*. The aim of the Candidate Enumeration step is to generate several regular expression patterns at different abstraction-level for an artifact, each of which becomes a candidate for the IOC for a specific family. Specifically, EIGER clusters the artifacts created by several malware samples belonging to a same malware family based on its string similarity, and then converts the set of similar strings in a same cluster to multiple regular expression patterns at different abstraction-level. The point is that EIGER carefully takes the level of abstraction of a regular expression into account and handles all of the patterns as an IOC candidate without discarding any of them at this time.

The aim of the Optimization step is to select accurate and interpretable IOC candidates as IOCs for the malware family from the ones generated in the previous step. In this step, EIGER optimizes a combination of IOC candidates through maximizing an objective function that takes both accuracy and interpretability into account. In particular, EIGER favors IOC candidates with high precision and high recall for accuracy. At the same time, it favors those with high conciseness, high coverage, and less overlap for interpretability. The point is that we solve this optimization problem as a non-monotone submodular function maximization problem where we can efficiently compute a near-optimal solution. Then, we convert the selected IOC candidates into the set of IOCs for the malware family. We iterate this series of the sub-steps for each malware family. Consequently, we can obtain a set of IOCs with the appropriate abstraction-level for multiple malware families which can be used in malware detection.

We implemented EIGER and the other components, such as *Dataset Creation*, *Format Conversion*, and *Incremental Learning*, in a prototype system for realizing a fully-automated IOC generation procedure. The system does not require any human interventions for generating IOCs from malware binary files.

We evaluated the performance of IOCs generated with the system from the viewpoints of accuracy and interpretability. As the former, we collected 162K malware samples over the recent five months, generated 1,326 IOCs (our IOCs) for 289 families from the samples, and compared the detection accuracy of our IOCs to other methods including machine learning (training and validation set size 89K, test set size 72K). This experimental results report that the performance of our IOCs is a True Positive Rate (TPR) of 91.8% and a False Positive Rate (FPR) of 0.97%. These numbers indicate that EIGER can produce IOCs whose detection accuracy is almost equivalent to other methods.

Additionally, we confirmed that Incremental Learning over five months can improve the TPR of IOCs up to 94.70% and fit the abstraction-level of the regular expression patterns within IOCs to new malware samples. From this result, we believe that EIGER and our IOCs are practical enough for SOC or CSIRT analysts to use for their daily tasks in order to automate parts of their tasks. We also confirmed that the precision and the FPR of our IOCs do not degrade for a completely different dataset obtained at different vantage points. The result demonstrates EIGER can complement commercial IOCs from a different point of view.

As the latter, we posit that manually-generated IOCs are sufficiently interpretable, and then we conducted a user study with security analysts to measure the difference between our IOCs and manually-generated ones in terms of user perception. We confirmed by statistical hypothesis testing that there was no statistically significant difference between the user perceptions of them. This experimental results demonstrate that EIGER can generate as interpretable IOCs as human-generated ones.

Contributions. Our contributions are as follows:

- We propose EIGER, a method to automatically generate state-of-the-practice IOCs from given malware traces. This method is built upon *enumerate-then-optimize* design which can be applicable to existing signature generation techniques in general.
- We implemented EIGER as a fully-automated IOC generation system and evaluated our IOCs using 162K samples showing a False Positive Rate (FPR) of 0.97% and a True Positive Rate (TPR) of 91.98%, which is comparable to known machine learning techniques. Additionally, we confirmed that EIGER can automatically refine the IOC through Incremental

FileItem/FullPath **matches** C:\\Users\\.*\\AppData\\Local\\Temp\\(aareg\.exe|install\.vbs)
OR RegistryItem/Path **contains** HKCU\\Software\\Microsoft\\Windows\\CurrentVersion\\Run\\WinUpdate64
description This is an IOC to detect artifacts generated by "Remcos" malware. The sources of this IOC are following: 96b799a7 ...

Figure 1: Part of the IOC signature that represents Remcos malware. Notation of OpenIOC [19] is simplified for easy viewing.

Learning and achieve high precision even for a completely different dataset.
- We show a first user study result on the interpretability of IOCs, demonstrating that there is no statistically significant difference between our IOCs and manually-generated IOCs. The result indicates that our IOCs can be seamlessly incorporated into the actual security operations.
- We make our IOCs publicly available[1] to promote further research in this area.

The rest of the paper is organized as follows. The background and goals we encountered are introduced in §2. The overview of our method and system is explained in §3. The details of Candidate Enumeration and Optimization are described in §4 and 5, respectively. The overview of our evaluation is introduced in §6. The experiments on accuracy and interpretability are detailed in §7 and 8, respectively. The limitations and the mitigations are discussed in §9. §10 reviews related work. Finally, we conclude our work in §11.

2 BACKGROUND AND MOTIVATION

In this section, we first explain the format of IOC and the terms used in this paper. Then, we define our goals and items outside the scope of the paper.

2.1 Indicator of Compromise

Format. Known IOC formats such as OpenIOC [19] and STIX [12] support operators expressing relationships between multiple artifacts and support multiple matching schemes. They support "AND" and "OR" operators and following matching schemes: "is" (complete match with a certain string), "contains" (whether a certain string is included in a given match target), "startswith" (whether the target start with a certain string), and "matches" (whether a regular expression matches the target). The target is typically a given log or a list per type of artifacts. In addition, we can fill in additional information as "description". Note that the expressiveness of different formats is not equivalent. Specifically, the AND operator in STIX can target any kind of artifact while that in OpenIOC has a limited scope [37]. In other words, STIX is more expressive than OpenIOC.

An example of IOC, generated by EIGER, is shown in Figure 1. This IOC indicates the infection of Remcos RAT[2][3] when either the file "aareg.exe" or "install.vbs" exists in the temporary directory, or the registry key "HKCU ..." and the value "WinUpdate64" exist in an endpoint.

Terms. We define the terms regarding IOCs we use throughout this paper. An IOC is a representation of one or multiple malware artifacts. We can use a string or regular expression for an IOC. When we use a regular expression as an IOC, we can cover multiple artifacts whose names are matched to the regular expression with the single IOC. Basically, we use a set of IOCs combined with the "OR" operator for a specific malware family. For example with Figure 1, "FileItem/FullPath ..." is an IOC, while "RegistryItem/..." is also an IOC. These IOCs, which are combined with "OR", are a set of IOCs for Remcos, a malware family. When we write down the set of IOCs into a file with adding the metadata, such as its description, we call the file an IOC or an IOC signature.

2.2 Motivation

Goals. To generate IOCs suitable for the use cases in an appropriate form, we pay particular attention to the two critical requirements: accuracy and interpretability. The mandatory requirement is that the IOC should contain sufficient amount of information to distinguish one malware family from benign programs or other malware families. In addition, the artifact and its logical combination described in the IOC must be a form that is easy for analysts to interpret. This is because the IOC use cases inevitably involve human interventions. For example, when we find the artifacts which are described in an IOC in an endpoint, we usually validate if the detection is related to malware infections or just a false positive. To make the verification process practical, the interpretability of IOCs is crucial because having interpretable IOCs enables us to quickly check if the items hit in the IOCs and start a further deep inspection for the identified artifacts.

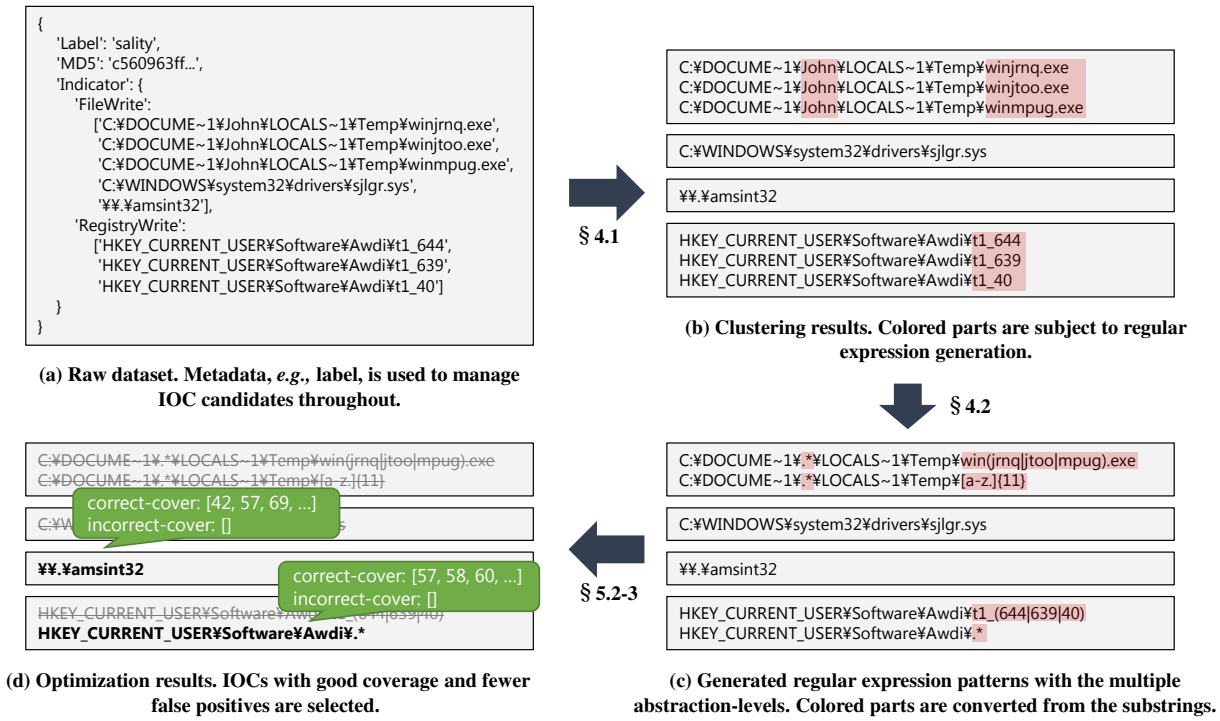
To satisfy these requirements, we define our goals as follows:

(1) **Accurate:** An artifact with a high recall and precision is expressed in the appropriate abstraction-level to classify the malware family; our goal is basically a multi-class classification as accurate as known machine learning methods.
(2) **Interpretable:** Reasonable information for analysts is described; there is no inferiority in user perception between EIGER-generated IOCs and manually-generated ones.

In summary, our goal is to generate interpretable IOCs whose detection accuracy is same as known machine learning methods.

We also have three additional goals to achieve with our system in which our proposal method and other components are implemented. First, we try to make our IOCs *vendor-agnostic*. That is, we can mutually convert our IOCs to equivalent representations for any IOC-based tool or product. Second, we design our system to generate IOCs based on coarse-grained behavioral information that we can collect from endpoints to avoid posing significant workloads on the endpoints.. Third, we design our system to keep up with the speed of rapidly emerging new malware variants. We consider that these additional goals/requirements are necessary to make our system practical and applicable to real-world use cases.

[1] https://github.com/malrev/eiger
[2] MD5: 96b799a78f8d48cb619a5e93b3191304
[3] MD5: 37d8dbdb9bf04326101a645e8e04da0d



(a) Raw dataset. Metadata, *e.g.,* label, is used to manage IOC candidates throughout.

(b) Clustering results. Colored parts are subject to regular expression generation.

(c) Generated regular expression patterns with the multiple abstraction-levels. Colored parts are converted from the substrings.

(d) Optimization results. IOCs with good coverage and fewer false positives are selected.

Figure 2: Overview of steps for generating IOC from malware trace log in (a). Figure 2(d) presents the completed version of indicator representations.

Items outside the scope of the paper. We note the following three items are out of scope of this paper. First, we do not aim to automate the procedure of the analyst analyzing the malware damage using the IOC. Second, we do not focus hash values of malware binary files although IOC formats allows us to describe hash values because the detection based on hash values provides less protection than static signatures. Third, we do not aim to create a static signature such as YARA [46], even though it is often mentioned as an instance of IOC. This is because our focus is on malware behavior. Yet, our method is potentially applicable to the generation of YARA.

3 OVERVIEW

In this section, we provide an informal overview of our proposal on an illustrative example. Figure 2 illustrates the steps of EIGER. Figure 2(a) shows a malware dataset. Only endpoint-obtainable features such as file path, registry key, *etc.* are present in the dataset. Given the artifacts in the dataset for each malware family, EIGER generates and selects IOCs shown in Figure 2(d). There, we can see that similar artifacts are aggregated as regular expression patterns. More interestingly, within the generated IOC, regular expression patterns matching hard-coded strings and regular expressions matching wider-range strings are combined in a manner to maximize both accuracy and interpretability.

We now explain the components of the system in due course. In particular, Candidate Enumeration and Optimization are an embodiment of the method based on our *enumerate-then-optimize* design, that is, EIGER.

Dataset creation. EIGER takes as input a dataset that consists of the trace log of malware and its label. Here, each attribute value of the trace needs to be a dictionary composed of the type of artifact and its entity. To generate a dataset, we first put malware samples into a sandbox and set a label for each sample. We then shape the analysis result as shown in Figure 2(a). This component can be incorporated with any sandbox or labeling method. For our method, the sandbox can be implemented via any operating system, any instrumentation method, and any network perimeter as long as it can acquire the artifacts to be IOC candidates. The labels can be obtained from any antivirus software or any analysts' decision as well. The dataset used for our evaluation is described later in §7.

Candidate enumeration. Given a malware dataset, EIGER *enumerates* regular expression patterns with different abstraction-levels as candidates for IOC. As shown in Figure 2(b), EIGER clusters similar attribute values for each malware family. Here, EIGER performs clustering separately for each type of artifact of attribute value. On top of that, EIGER generates regular expression patterns for each hierarchy of similar artifacts in each cluster as candidates of the part of IOC.

As shown in Figure 2(c), patterns are generated with up to three abstraction-levels: A pattern made from a hard-coded string of the artifacts *i.e.,* "win(jrnq|jtoo|mpug).exe", a pattern made from character type and its range *i.e.,* "[a-z]{11}", and a pattern matches arbitrary string *i.e.,* ".*". If the cluster contains only a single artifact, no regular expression is generated and the artifact becomes

a candidate directly. Note that artifacts that are clearly variable by environment such as user name are converted to ".*" in advance.

The generated IOC candidates are associated with each malware family and thus can be regarded as rules for multi-class classification. We detail this step later in §4.

Optimization. After the IOC candidates are enumerated, EIGER derives their subset through *optimizing* their combination for maximizing the accuracy and the interpretability. In our example, EIGER selects the IOCs in Figure 2(d) from the IOC candidates in Figure 2(c). To this end, we introduce an objective function that scores both accuracy and interpretability, based on Interpretable Decision Sets (IDS) framework [32].

IDS uses an instantiation of non-normal submodular function maximization to optimize set of rules for a dataset *i.e.,* rule learning. This framework is applicable to IOCs, as it supports rules of any structure if it includes labels and attributes. The key idea of IDS is to select only concise and accurate rules from a given set of rules, so as to cover the entire data set with as few rules as possible. This is based on the insight that concise and high coverage rules are more "interpretable" than others. Fortunately, in the original paper, the validity of the insight has been demonstrated by a user study comparing IDS-derived rules with rules derived from other machine learning methods. We focus on this point and aim to verify whether the IOCs optimized by IDS are interpretable to the same extent as the IOCs optimized by analysts. Although this optimization task belongs to the NP-hard class, properties of such a submodular function enable us to guarantee a $2/5$ near-optimal solution [18].

We have improved the objective function towards our domain-specific task. Our improvements include the redefinition of the metric of conciseness and the solution to a class imbalance problem. This step and our insight are described in §5.

Format conversion. The IOCs which have been optimized by EIGER thus far are pseudo-IOC in format. Therefore, they cannot be deployed as a concrete signature at this point. Thus, through this component, our system converts the optimized set of IOCs *e.g.,* Figure 2(d) to the IOC signatures applicable to real-world products. Specifically, we produce IOCs as OpenIOC [19] and also as STIX [12]. In order to produce IOC signatures in a vendor-agnostic form, our system outputs the final IOC signature in OpenIOC format because it is the simplest IOC form. There, IOCs of the same family are OR-ed together and one IOC signature is output for each family. During this process, our system adds the hash value of samples and the search link of external sources to the description of the IOC signature to facilitate the analysts' operation.

Incremental learning. This component works as a runner of the aforementioned components to track the emergence of new malware. As attackers keep working to improve malware daily, IOCs that perform well at certain times may not work forever. Therefore, if there is a change in the malware family, it will be necessary to make changes to the IOCs corresponding to that family. The same is true if a new family appears. Conventionally, such an operation that improve the IOC to track the new malware emerges have been done in manuals.

We address this problem by optimizing previously-generated IOC candidates for a most recent dataset. Suppose there are datasets made over time for a certain time window. Here, when optimizing IOCs for a most recent dataset, instead of running Optimization step for only candidates generated from the most recent dataset, we run the step for all candidates which are cumulatively generated from the past datasets. This enables us to select near-optimal IOCs for the current dataset from all previous candidates and to delete the IOCs that no longer work. The efficacy of this component is described in 7.

4 CANDIDATE ENUMERATION

In this section, we describe how Candidate Enumeration step, introduced in §3, *enumerates* multiple regular expression patterns with different abstraction-levels. To generate regular expressions, EIGER executes two sub-steps. First, EIGER clusters the artifacts included in the dataset for each malware family based on the string similarities. Second, EIGER converts the artifacts contained in each cluster into a trie data structure [3] and generates multiple regular expression patterns. In the remainder of this section, we provide a description for each sub-step.

4.1 Clustering

The aim of this sub-step is to put together similar artifacts into a cluster. Given a dataset, for each malware family, this sub-step clusters the artifacts with the single linkage method [26].

Given a data for each family of N artifacts, we first create an initial state with N clusters containing only one artifact. Starting from this state, the distance between the clusters $d(C_1,C_2)$, where C_1 and C_2 is a cluster, is calculated from the distance $d(x_1,x_2)$ between x_1 and x_2 where x_i is an artifact, and the two clusters closest to this distance are merged sequentially:

$$d(C_1,C_2) = \min_{x_1 \in C_1,\, x_2 \in C_2} d(x_1,x_2) \tag{1}$$

where the function d computes Levenshtein distance, that is, the distance that takes artifacts as a string.

The reason why we adopted the single linkage method is that we were able to split clusters with finer granularity than other clustering methods *i.e.,* DBSCAN [16] and X-means [43] in pilot testing. Note that our aim here is to cluster the candidate set to be put together into a regular expression pattern, therefore the cluster should be as small as possible.

4.2 Regular Expression Generation

The aim of this sub-step is to generate regular expressions that become IOC candidates from the clustering result. Given a cluster, this sub-step first divides the artifacts contained in the cluster into each hierarchy and creates a list of substrings that were in the same hierarchy. The hierarchy referred to here is composed of substrings of artifacts divided by a backslash in a path of Windows, and substrings of artifacts divided by octets in an IP address. Then, EIGER converts multiple substrings contained in the list to a regular expression. Finally, EIGER outputs strings combining regular expression patterns or substrings for each hierarchy as final candidates. At this time, if there is only one substring in the list of substrings for each hierarchy, the substring is not converted. Also, user names are converted to ". *" beforehand.

During this process, EIGER generates regular expression patterns up to three abstraction-levels: hard-coded pattern, character class

with range pattern and arbitrary string pattern. Hard-coded patterns are always generated, and character and its range patterns and arbitrary string patterns are generated as needed. The key insight here is much malware may create artifacts that contain specific strings, but some of them may create pseudo-random filenames, from a return value of a certain API *e.g.,* GetTempFileName and GetPerformanceCounter. Therefore, hard-coded patterns alone are not sufficient for malware detection (in other words, the generalization ability is low), while character and its length patterns and arbitrary string patterns are not suitable for all malware. Thus, EIGER first generates a hard-coded pattern, and only if the artifacts seems to be pseudo-random filenames, EIGER tries to generate a character and its range pattern, otherwise EIGER generates an arbitrary string pattern. We describe the method of generating each pattern below.

Hard-coded pattern. EIGER first generates regular expression patterns with the coarse abstraction-level. Here EIGER combines the list of substrings into a single trie structure and regularizes it. Trie is a tree where each node represents a single character or a prefix. The root is an empty string. A node that are k edges of a distance of the root have an associated prefix of length k.

For example, let us consider a list containing just two words: "ACSAC" and "ACCEPT". Given the list, "" → "A" → "C" would be generated. The node "C" would have two children: "S" and "C". Each node grows "S" → "A" → "C" and "C" → "E" → "P" → "T", respectively. Once a trie generated, EIGER converts the branch to a token "|" and converts the trie to a single pattern. Strings in the different branch from the same node are enclosed in grouping brackets "()". In that way, we generate "AC(SAC|CEPT)" from "ACSAC" and "ACCEPT".

This is the most concrete pattern that EIGER generates and that previous works [40, 44, 55] do not generate. In order to avoid degrading the interpretability, if the regular expression branch is likely to be nested, EIGER separates it as a part of another regular expression pattern. Also, if there is only a single artifact in the cluster, EIGER will not create a regular expression but will add that artifact to the candidates directly.

Character class with range pattern. If the length of strings in the list is, all the same, EIGER generate a regular expression that represents the type and range of the characters in that string. For example, EIGER does not generate this pattern from "ACSAC" and "ACCEPT", but generates "[A-Z]{5}" from "ACCEPT" and "ACCESS".

Arbitrary string pattern. If the list contains strings with different length, EIGER generates a regular expression that matches arbitrary string. For example EIGER generates ".*" from "ACSAC" and "ACCEPT", but does not generate this pattern from lists contains two substring "ACCEPT" and "ACCESS".

EIGER applies these regular expression generation techniques to each hierarchy of the artifacts in the cluster. Since EIGER performs these regular expressions for each hierarchy of the artifact, the maximum number of candidates generated for each cluster is the power of the number of the abstraction-levels (three) and the hierarchy subject to regular expression for each cluster. Intuitively, hard-coded patterns seem to be under-approximation of the artifacts that the malware family may potentially generate, and arbitrary string patterns are its over-approximation. However, in fact, it cannot be determined whether a candidate could appropriately approximate artifacts or not without measuring its coverage for malware samples. Therefore, in the next step, EIGER will match the dataset with the IOC candidates and choose IOCs that gives the appropriate approximation.

5 OPTIMIZATION

This step *optimizes* a combination of the IOC candidates generated in the previous step to derive a set of IOCs which approximate potential malware artifacts. That is, the optimized IOCs are the subset of the candidates. To this end, we adapt IDS [32], a rule learning framework based on a theoretical foundation called the submodular function maximization.

Given a dataset D, a set of itemsets S, and a set of possible class labels C, IDS can find a near-optimal rule set (subset $S \times C$ of itemset and class label pairs) that makes accurate predictions with clearly described decision boundaries. IDS is applicable to any itemset and class label pairs, and in our usage, the IOCs and the indicated families corresponds to $S \times C$. The performance of IDS is comparable to other machine learning techniques and the decision boundary of it is "interpretable" than others (the user study of the interpretability aimed at comparison with other machine learning techniques is included in the original paper). Therefore, employing this framework is reasonable to achieve our goals.

In the remainder of this section, we first introduce the submodular function maximization, and we then explain IDS. Following these preliminaries, we describe our extension to the original IDS and our insight.

5.1 Submodular Function Maximization

Formally speaking, a function $f : 2^D \to \mathbb{R}$ defined on some set D is submodular if for any $A \subseteq B \subseteq D$ and $x \in (D - B)$, the following condition is satisfied:

$$f(A \cup \{x\}) - f(A) \geq f(B \cup \{x\}) - f(B) \tag{2}$$

To give an example, let us consider a coverage function for malware analysis results. We assume the function f which receives a set of IOCs and which returns the proportion of malware that matches the IOCs in the given set within the dataset. In this case, the number of malware that can be covered by f when a new IOC is added to the set is larger if only a few IOCs are already selected. A function with such property is referred to as submodular.

Although maximizing a submodular function is NP−hard [28], an approximate solution can be obtained efficiently. Intuitively, in the example, what we need to do is greedily adding the item with the largest coverage at the time for each iteration. If a function is non-negative, monotone and submodular, a greedy method gives 0.63−approximation [41]; if a function is non-negative, non-monotone and submodular, Smooth Local Search (SLS) gives 0.40−approximation [18]. Both of them accept candidate sets and return near-optimal subsets.

As a strong benefit in practical use, a non-negative linear combination of submodular functions $\sum_i \lambda_i f_i$ is also submodular where the weights $\lambda_i \geq 0$ by definition. This means maximization of multiple submodular functions can be executable at once.

Table 1: Objective functions of Interpretable Decision Sets [32] and our modification.

Function	Description
$f_1(R) = \lvert S \rvert - size(R)$	Represent the entire dataset with a few rules.
$f_2(R) = \sum_{r \in R} (\max_{r \in S \times C} width(r) - width(r))$	Highly evaluate a concise rule *i.e.,* rule with a few representations. We redefined $width(r)$ to the number of regular expression objects in r.
$f_3(R) = \sum_{\substack{r_i, r_j \in R \\ i \leq j \\ c_i = c_j}} (N - \lvert overlap(r_i, r_j) \rvert)$	Reduce the intra-class *overlap* of rules. N is the number of points in the dataset.
$f_4(R) = \sum_{\substack{r_i, r_j \in R \\ i \leq j \\ c_i \neq c_j}} (N - \lvert overlap(r_i, r_j) \rvert)$	Reduce the inter-class *overlap* of rules. We removed this function from our objective.
$f_5(R) = \lvert C \rvert - \sum_{c' \in C} \mathbb{1}(\exists r = (s,c) \in R \text{ such that } c = c')$	Have at least one rule for each class. We removed this function from our objective. The function $\mathbb{1}$ returns 1 whenever the condition it takes as an argument is true and 0 otherwise.
$f_6(R) = N \cdot \lvert S \rvert - \sum_{r \in R} \lvert incorrect\text{-}cover(r) \rvert$	To encourage precision, reduce the size of *incorrect-cover* sets.
$f_7(R) = N - \sum_{(\mathbf{x}, y) \in D} \mathbb{1}(\lvert \{r \lvert (\mathbf{x}, y) \in correct\text{-}cover(r)\} \rvert \geq 1)$	To encourage recall, have at least one accurate rule describes each data points.

5.2 Interpretable Decision Sets

Notation. We here introduce the notation of original IDS. Let $D = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$ be a dataset where each data point $\mathbf{x}_i$ is a tuple of attribute values $(x_1, \ldots, x_k)$ and y_i is the corresponding label. We denote by S a input set of itemsets s, C a set of class labels c in D, and $R = \{(s_1, c_1), \ldots, (s_k, c_k)\}$ a set of rules, respectively. We call an itemset-class pair (s,c) a rule r. In our usage, a rule corresponds to each IOC and an itemset corresponds to a representation of artifacts *i.e.,* an IOC.

Here, we consider $size(R)$ *i.e.,* the number of rules in R , $width(r)$ *i.e.,* the number of representations in the itemset s, and $cover(r)$ *i.e.,* the set of data points $\mathbf{x}$ in D that matches all the representations of an itemset s. To measure recall and precision of each rule, we also consider the following:

$$overlap(r, r') = cover(r) \cap cover(r') \quad (3)$$

$$correct\text{-}cover(r) = \{(\mathbf{x}, y) \in cover(r) \mid y = c\} \quad (4)$$

$$incorrect\text{-}cover(r) = cover(r) \setminus correct\text{-}cover(r) \quad (5)$$

Objective. Following these notations, IDS tries to maximize the combination of 7 objective functions at once:

$$\underset{R \subseteq S \times C}{\operatorname{argmax}} \sum_{i=1}^{7} \lambda_i f_i(R) \quad (6)$$

Each function is described in Table 1. The hyperparameters λ_1, λ_7 are set at initialization of the optimization, and an appropriate value is tuned by means like gird search. Given a set R containing all rules, IDS returns R' which contains only the necessary r through optimizing the objective. The goal of the objective is to accurately describe the entire dataset with a simple and concise set of rules. In particular, the functions $f_1, \ldots, f_4$ are for interpretability, the others are for accuracy. The proof that $\sum_{i=1}^{7} \lambda_i f_i(R)$ satisfies non-monotone, non-negative and submodular is described in a technical report version of IDS proposal [33]. Because the combination of these objective functions is non-monotone, IDS employ SLS for its optimization. As a result, IDS can derive the set of rules which is at least $2/5$ optimal solution (and the derived rules work sufficiently in practice).

Classification. To use IDS-optimized rules for classification, we follow these steps: Given attribute values x, if x matches exactly one itemset s_i, its class label is assigned to the corresponding c_i. If x matches no itemsets then its class label is assigned to a default label, and if x matches more than one itemset, it is assigned to a class through a majority vote. In our usage, we can classify malware by if we find the artifacts represented by an IOC.

5.3 Domain-Specific Extension

We redefined $width(r)$ in f_2 and we removed f_4 and f_5 for our IOC optimization. We describe each reason and our insight.

Conciseness metric. As we show in Table 1, the function f_2 is defined to minimize the clause number $width(r)$ of each rule in a given set. Unfortunately, this metric does not work for our IOC candidates. In the original IDS design, itemset s is extracted by frequent itemset mining [1]. Therefore, itemset may contains multiple artifact representations and $width(r)$ is defined as the number of representations. On the other hand, "itemset" we generate in Candidate Enumeration contains only one representation. Thus, $width(r)$ is identical for any IOC.

For reasons stated, we changed the definition of $width(r)$ from the number of representations in the itemset s to the number of regular expression objects in the "itemset" s. In other words, f_2 turns into a function that minimizes the total number of regular

expression objects in each rule in the set of rules. This modification is based on the insight that IOCs with fewer regular expressions are more interpretable. We can say that this function satisfies submodularity as before, following in the Appendix proof in [33].

Family-awareness. The function f_4 is defined to reduce the inter-class overlap of rules and f_5 is a function to have at least one rule per class. The commonality of them is that they are defined to optimize rule sets that contain multiple classes. Unfortunately, a class imbalance problem forced us to consider the reason for this function. This is the problem that a classifier which ignores minority classes tend to be trained when the number of one class of data is far less than the number of another class of data. This problem is common in the malware domain. For example, in the Drebin dataset [2], the largest family contains 925 samples, whereas 58% families consist of fewer than five samples. The original IDS did not consider this problem and if we set the objective function for the entire dataset, the rule that functions effectively for a small number of families will have a lower evaluation value compared to the rule for majority families. Hence, except for the rule protected by f_5, rules may be excessively reduced. In other words, rules for minority families will be understated.

For this reason, we divided the process of Optimization for each class. In line with this, we removed the function f_4 from our objective and decided to try optimization independently for each class. As a consequence, the function f_5 is no longer needed. This will neglect near-optimality of rules for the entire dataset, but will respect near-optimality of rules for each family. Note that *incorrect-cover etc.* are calculated using the entire dataset because we need to delete rules that misclassify other families.

6 EVALUATION OVERVIEW

In this section, we provide an overview of our series of experiments. Because our goal is to produce accurate and interpretable IOCs, we evaluate generated those from both perspectives. Especially, we design and run experiments to answer the following research questions:

(1) **RQ1:** Are EIGER-generated IOCs able to reach the same predictive *accuracy* as other methods for a labeled dataset?
(2) **RQ2:** Is EIGER able to improve the *accuracy* of IOCs when assuming one-week period to refine the IOCs?
(3) **RQ3:** Are EIGER-generated IOCs able to provide *accuracy* comparable to commercial ones?
(4) **RQ4:** Is EIGER able to generate IOCs that achieve the same *interpretability* as manually-generated ones?

As the response to RQ1, we compare an accuracy of our IOCs with accuracies of classifiers trained with the existing machine learning methods. The aim of RQ2 and RQ3 is similar to RQ1 in that they relate to accuracy, but with more practicability taken into account. In response to RQ3, by activating Incremental Learning component, we repeatedly refined IOCs from a dataset from the previous week and repeatedly measured the accuracy for the dataset of the next week. To answer to RQ4, we deployed a certain commercial EDR product which accepts IOCs and provides detection scheme, put the EIGER-generated IOCs into the product, and evaluated the performance with the pre-installed commercial IOCs. We detail this series of experiments in §7.

Table 2: Dataset.

		# classes	# samples	Date range
Train	Malware	289	80,239	Nov 2018 -
	Benign	1	9,664	Feb 2019
Test	Malware	287	63,993	Mar 2019 -
	Benign	1	8,130	Apr 2019

As the answer to RQ4, as we posit that manually-generated IOCs are interpretable, we presented our IOCs and manually-generated ones to SOC analysts and analyzed the difference in impressions from both. We describe this experiment in §8.

For these experiments, we instantiated EIGER in our system written in Python 3.6.5 built top on an Intel(R) Xeon(R) computer with an E5-2660 v3 CPU 16 cores and 192 GB RAM. Since our method can be executable separately for each family, we execute threads per family in parallel. This enables us to generate IOCs faster than sequential processing.

7 ACCURACY EXPERIMENTS

In this section, we detail the experiments to answer to RQ1, RQ2, and RQ3, respectively. The experiments were performed in accordance with well-established guideline [50].

7.1 Comparison with Machine Learning

Purpose. The purpose of this experiment is to compare the classification performance of EIGER-generated IOCs with that of other methods. In particular, we set up our rivals in two directions to evaluate our design choice. First is the IOCs from EIGER with weaker settings, that is, the ones optimized in the setting excluding the family-awareness mentioned in §5 and the ones optimized without generating regular expression patterns. Second is the other methods that purely seek accuracy without considering interpretability *i.e.,* machine learning.

To these ends, we divided 162K samples with ground-truth labels into training set and test set and measured the predictive performance of each method. This experiment follows a general way to evaluate machine learning methods. Note that our purpose is to measure the performances on coarse-grained and endpoint-obtainable artifacts and thus it is not to compete against state-of-the-art which is hinging on fine-grained information obtainable through dedicated instrumentation *e.g.,* data flow analysis [58] and multi-path execution [10].

Dataset. Table 2 summarizes the dataset used in this experiment. The malware samples were collected from VirusTotal [35] from November 2018 to April 2019. All malware samples were Windows PE file format, detected by 19 or more but 29 or less anti-viruses in VirusTotal. The benign samples were collected from public sources of free Windows software. To generate the dataset, we adopted Windows 7 as an operating system, adopted the out-of-the-box taint analysis *i.e.,* API Chaser [27] as a hook method to stealthy track successful write events to the file system or registry and command line arguments to accompany process creation. We also carefully filtered the network communication to prevent from attacking third

Table 3: Classification results on the test set. The TPR is slightly inferior to the machine learning methods, but the FPR and the precision are superior to others.

Method	FPR%	TPR%	Precision%	F1 score
EIGER	**0.97**	91.98	**99.87**	95.76
$EIGER_{D-}$	4.76	90.34	99.34	94.62
$EIGER_{R-}$	6.03	70.20	98.92	82.12
DNN	1.00	**93.82**	**99.87**	**96.75**
XGBoost	2.20	91.31	99.69	95.32
Random Forest	1.19	93.59	99.84	96.61
Decision Tree	1.23	92.73	99.83	96.15

parties. The analysis time of each sample was set to 30 minutes. Our ground-truth labels *i.e.,* family names for each malware sample were obtained from AVCLASS [53], a tool which takes as input VirusTotal reports and returns a label that aggregates scan results of multiple anti-viruses.

Procedure. We used the first three months of the dataset for IOC generation or model training and the second two months for their testing. This split ensures that future knowledge about the testing objects does not enter the training phase [17], that is, we have properly assessed the accuracies with preventing data leakage.

We generated IOCs from the training set using EIGER, $EIGER_{D-}$, and $EIGER_{R-}$. EIGER is our proposal, $EIGER_{D-}$ is EIGER without domain-specific extension, and $EIGER_{R-}$ is EIGER without creating regular expression patterns. We also trained Deep Neural Network[4], XGBoost [7], Random Forest [4], and Decision Tree [5] from the training set. We here used bag-of-words feature extraction, referring to Rieck *et al.* [49]. We performed 5-fold cross validation during IOC generation or training to adjust hyperparameters, such as $\lambda_1 ... \lambda_5$ in our method and number of trees in Random Forest, through the grid search. In other words, we used part of the training set as a validation set. After the IOC generation and the model training, we measured multi-class classification performance on the test set.

Results. EIGER successfully generated IOCs in an average of 1,710.53 seconds (28.5 minutes) during the cross validation. The best TPR of the EIGER-generated IOCs on the training set was 97.53%. Eventually, 1,326 IOCs were generated.

The results on the test set shown in Table 3. From the table, we observe the following. First, the EIGER-generated IOCs successfully detected malware as accurate as known machine learning methods. This implies we can represent most malware with a discrete pattern which does not lie in multiple artifacts. Second, we can suppress FPR with the extension considering the class imbalance problem (see EIGER and $EIGER_{D-}$). Third, we can improve TPR about 20% by introducing regular expression to an IOC (see EIGER and $EIGER_{R-}$). We consider the reason why FPR of EIGER is lower than other methods is because malware has reduced to the presence or absence of a single artifact. The results enables us to conclude that EIGER has enough capability to generate accurate IOCs.

[4]The number of the layers was three , the activation function was ReLU [39], and the optimization method was Adam [30]. 20% dropout.

Table 4: Snipped classification results on top-wanted malware [20] in the test set. Among them, Mirai is excluded because it is targeting Linux; Pushdo is excluded because it did not exist in our dataset.

Family	TPR%	Precision%	F1 score	# samples
Emotet	100.00	100.00	100.00	1138
WannaCry	100.00	88.00	93.1	863
Kovter	100.00	100.00	100.00	1305
ZeuS	100.00	97.27	98.50	90
Dridex	100.00	100.00	100.00	131
IcedID	100.00	100.00	100.00	105
Gh0st	100.00	100.00	100.00	812
NanoCore	100.00	94.82	97.51	24

Table 5: Classification results on the final week's part.

Method	FPR%	TPR%	Precision%	F1 score
$EIGER_{I+}$	**0.94**	**94.70**	**99.90**	**97.10**
EIGER	0.97	91.98	99.63	95.52

Additionally, for readers who may be interested in how much EIGER works against the serious threats, Table 4 shows a snipped classification result on the top-wanted malware [20]. The result indicates that EIGER has a potential to cope with not only malware in general but also serious malware.

7.2 Effectiveness of Incremental Learning

Purpose. The purpose of this experiment is to measure the performance of EIGER in a bit more realistic scenario. The IOC is, after all, a signature, so we need continuous updates. Therefore, we here run an experiment in a scenario that incorporates IOC updates with activating Incremental Learning. Through this experiment, we can see how Incremental Learning can improve the IOC's performance.

Procedure. We use the same dataset used in the previous experiment. We compared $EIGER_{I+}$-generated IOCs and EIGER-generated ones. The former is IOCs which improved weekly with Incremental Learning activated and the latter is original IOCs from the first three months of the dataset (the same IOCs generated in the previous experiment). For $EIGER_{I+}$, we repeated the following: we first generate IOCs from part of the dataset until a certain period and then measure the performance with next week's part. We did not refine the EIGER-generated IOCs, but measured the performance repeatedly.

Results. The weekly performance trend is shown in Figure 3, and the snipped result of the final week is shown in Table 5. Unsurprisingly, $EIGER_{I+}$ finally outperformed EIGER. From the figure, we observe the following. First, our Incremental Learning mainly contributes to recall rather than precision. Second, the larger the test set, the lower the TPR, but the performance will converge without the degradation if at least new samples are around 5,000 per day.

We also confirmed how Incremental Learning can automate IOC refinement. An example of IOC refined by Incremental Learning is

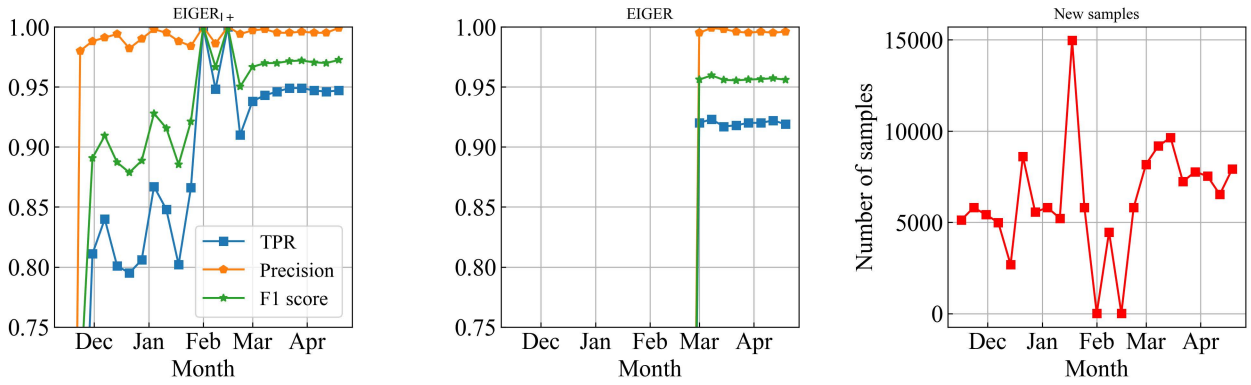


Figure 3: Weekly results of EIGER with incremental learning and without. Note that the daily sample size is different.

Table 6: Classification results under the adverse condition.

Method	FPR%	TPR%	Precision%	F1 score
EIGER	**0.18**	**12.25**	**98.28**	**21.59**
Product A	4.31	7.24	95.68	13.23

against malware Dofoil[5]. This malware leaves registry keys including GUIDs, pseudo-randomly named files, and command line arguments containing specific strings like "wind.sfx.exe" as artifacts. The last is the only artifact can truly be used as an IOC, but we first chose a regular expression pattern converted from the registry keys from its coverage. However, as learning proceed, it turned out that this pattern falsely detects benign samples. Therefore, EIGER had started selecting the command line arguments to suppress FPR for benign sample, willing to drop TPR for Dofoil a bit (from 90% to 88%).

These results suggest that our system frees analysts from the procedure of maintaining IOCs.

7.3 Comparison with Commercial IOCs

Purpose. The purpose of this experiment is to analyze the difference between our IOCs and commercial ones. Many commercial products used for Endpoint Detection and Response (EDR) services offering by SOC attempt to detect malware using pre-installed IOCs in combination with IOCs added by the user. Thus, we can compare our IOCs and commercial ones through deploying an EDR product and adding our IOCs to it. Due to resource constraints (in the previous evaluations, we could measure the performance if only traces obtained once, but in this evaluation, we need to actually deploy the product separately from the analysis environment), we cannot use the same test set of the dataset in Table 2 in this experiment. Thus, we set up a completely new test set for this comparison. This was inevitable, but it is also interesting because malware can be generated from a completely different distribution from the training set. Therefore, if the result is good, it is suggested that EIGER will thrive even under such an adverse condition.

[5]MD5: d82d84ed60c6b121bfa7d117538c9fcb

Dataset. We collected samples posted to Hybrid Analysis [14], an online sandbox service which makes a verdict whether a submitted sample is malicious or not. The sample collection period is April of 2019, and the total number of samples is 1,747.

We set the label according to the following procedure: Put the sample into our on-premises Cuckoo Sandbox [21], a malware scoring sandbox instance, make the label malicious if Hybrid Analysis's verdict is malicious and Cuckoo's malicious score 2.5 or more, make the label benign if Hybrid Analysis's verdict is others and Cuckoo's malicious score 2.5 less. Consequently, we had 808 benign samples and 939 malicious samples. In other words, the task here is binary classification.

There are two reasons for this labeling. The first reason is that the samples do not overlap with VirusTotal and we cannot configure ground-truth labels. The second reason is that the EDR product neglects malware classes and the commercial IOCs can only be used for binary classification. Thus, while our IOC describes malware families, but this time we ignore the families and conduct binary classification (malicious or benign).

Procedure. We installed the product on a Windows 7 machine. The EDR product we used had at least 20 IOCs for generic malware detection pre-installed (total number is private). Next, we put the IOCs generated by EIGER from the training set of Table 2 (same ones used in the previous experiments) into the product (the IOCs generated from the benign samples were excluded). After that, we run new daily samples in the environment every day to determine if the IOCs would react.

Results. Figure 4 and Table 6 shows the results. From the figure, we observed the following. First, TPR for different data sets had reduced, unfortunately, but unsurprisingly. Second, our IOCs maintain high precision and law FPR even on the different test set.

In addition, the number of samples that our IOCs could detect but the commercial ones could not detect was 113 out of 190 total detection. Conversely, the commercial ones could be detected, and the number of samples ours could not detect was 57. Therefore, regarding this less overlap in the detection, we can say EIGER complements commercial IOCs.

The experimental results suggest that EIGER is practical in its precision even under an adverse condition. In combination with the

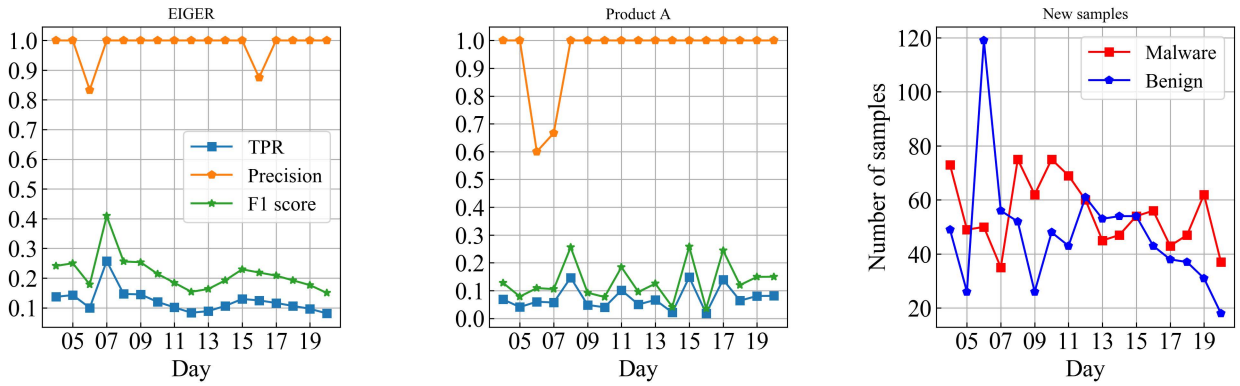


Figure 4: Daily results of EIGER and pre-installed commercial IOCs. Note that the daily sample size is different.

previous evaluation results, we consider proper sample collection can further improve the practicality of EIGER.

8 INTERPRETABILITY EXPERIMENT

In this section, we present the experimental results to answer to RQ2. To the best of our knowledge, this is the first user study that explores the interpretability in the malware detection field.

8.1 User Study

Purpose. The purpose of this experiment is to measure the interpretability of our IOCs in a realistic context. As we discussed in §1, the IOC plays a vital role in the security measure against malware threats. Because security analysts need to inspect the validity of an IOC when it triggers an alert, it is crucial that the generated IOCs are interpretable for the analysts. Our insight is that existing IOCs are created by analysts and are hence interpretable to analysts. Therefore, we can discuss how our IOCs are interpretable by comparing the IOC signatures created by the analysts with ours. Thus, we planned a user study in which participants have to answer the perception from given EIGER-generated or manually-generated IOC signatures.

Participants. We recruited 15 analysts (14 male, one female) working for a SOC in Japan, which is providing managed security service. Their job role includes monitoring, triage, analyzing and reporting alarms from network devices and endpoints. Since they have the opportunity to interact with IOCs on a daily basis, they are suited for evaluation in line with the use case. The author of the manually-generated IOC signatures belongs to the SOC but was not included in the participants of this experiment.

Among the group, the median age was 31. The oldest participant was 50 years old and the youngest was 27 years old. Their experiences in the security field were 5.26 years on mean, five years on median. We told them that we wanted to conduct a study on IOC creation but not mentioned its automation in order to minimize the risk of bias. To motivate them to join an experiment, we offer them to provide our IOCs after the experiment.

Since we cannot reach out the entire SOCs in the world, this experiment is limited in scope. Even though, we believe that we can validate the interpretability of IOC signatures for general SOC analysts in this way because they have a commonality in their workflow.

Design. We designed a user study where each participant was presented with EIGER-generated IOCs and manually-generated IOCs, through an online platform. Specifically, we presented three EIGER-generated IOC signatures and three manually-generated ones to the participants as OpenIOC [19] format with its creator was hidden. For the former, we randomly took the IOC signatures from the generated set of IOCs and for the latter, we used the ones given by its author. The author, a SOC analyst, was known to us but was not involved in the study or the work on the automated IOC generation. Note that the IOCs used here are IOC signatures. Each one includes a description, multiple operators, and representations as shown in Figure 1. As a difference from ours, all the descriptions in the manually-generated those are precisely-written in natural language.

We showed questions for each IOC signature in which participants rate their level of agreement about its interpretability. The presented questions are as follows: *(1) Do you think that you correctly understood what this IOC means? (2) Is this IOC easily readable? (3) Is this IOC well-structured? (4) Is this IOC seamlessly usable in your daily work? (5) Does this IOC look similar to the way you would create IOC? (6) Is this IOC high quality? (7) Is this IOC reliable? (8) Is this IOC sufficient for the operation even without the raw log?* These were inspired by the previous work on a usability-aware decompiler [59].

The ratings are 5-point Likert scales, that is, the choices are: *Strongly Disagree*, *Disagree*, *Neutral*, *Agree*, and *Strongly Agree*. Note that each IOC signature was presented without being told its creator (EIGER or the analyst), so we can expect an unbiased evaluation of it. Participants were allowed to use the Internet during the experiment.

In order to obtain the deep insights into the reasons why participants reported in that way, participants were asked *What elements do you want in terms of logic or description of IOC in addition to the presented ones?* as an optional descriptive question after finishing the choices.

Results. Figure 5 summarizes the ratings. Both ours and the manually-generated IOCs scored *Agree* on mean. In EIGER, 14% of the agreements was *Strongly Agree* or *Disagree*, 31% was *Neutral*, 56% was *Agree* or *Strongly Agree*; In Manual 15% of agreements was

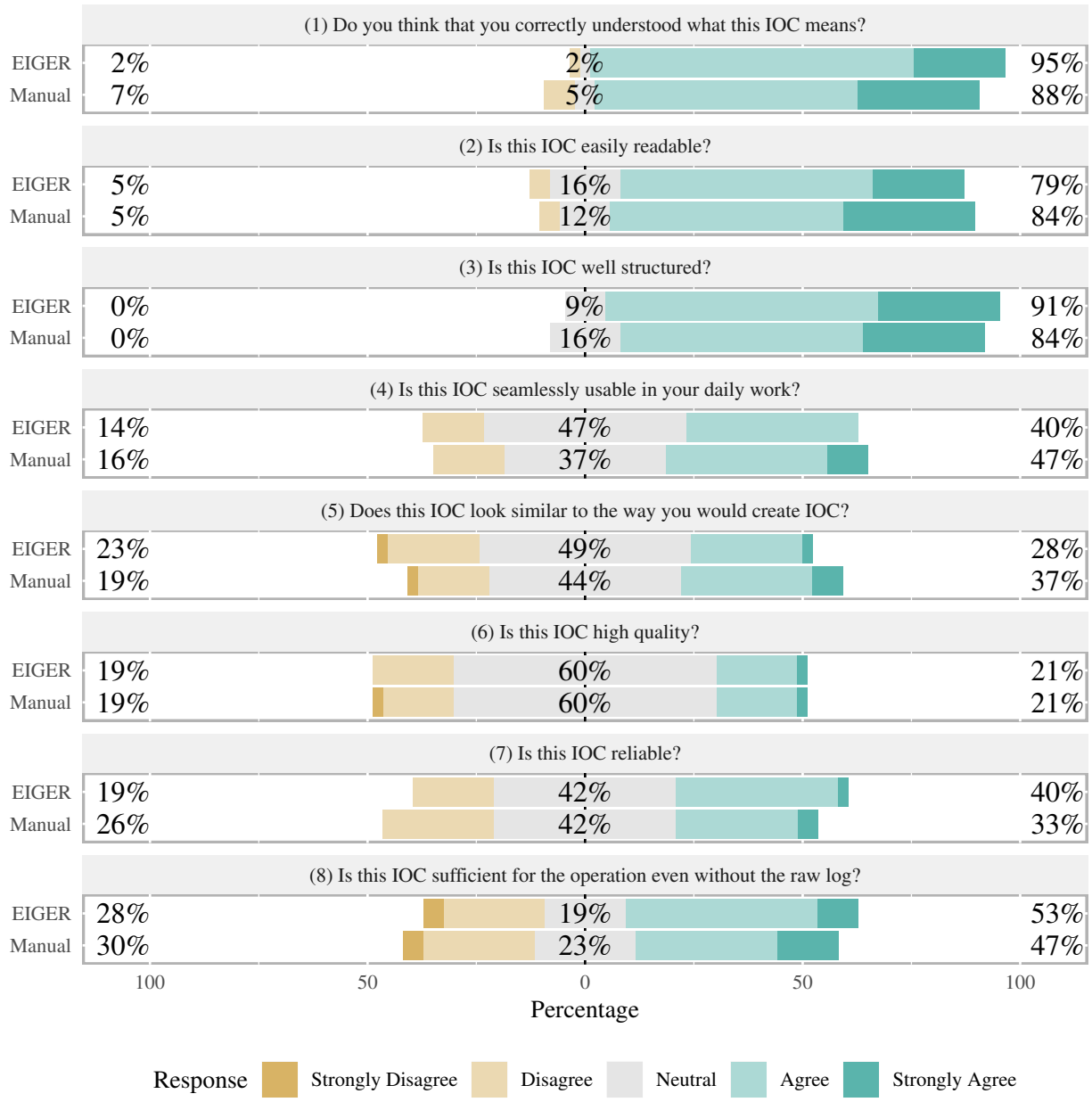


Figure 5: Participants ratings to the questions. Our IOCs scored slightly higher than manually-generated ones in (1), (3), (7), and (8). In contrast, our IOCs scored slightly lower than manually-generated ones in (4) and (5). Even though, there is no statistically significant difference between them ($p = 0.55$).

Strongly Disagree and *Disagree*, 30% was *Neutral*, 55% was *Agree* and *Strongly Agree*. According to the difference of the timestamp between GET request and POST request of the online platform, the response time to the experiment was 42.28 minutes on median and 48.55 minutes on mean.

In order to confirm the statistical difference between the score of our IOCs and the score of the manually-generated ones, we adopted the Wilcoxon signed-rank test [57]. The reason why we took this test is because of the two reasons. First, our subject here is an ordinal scale. Second, the results did not follow the normal distribution[6] and thus t-test is not suitable [52]. The p-value was 0.55 which indicates that there is no significant difference in user perception between ours and manually-generated ones. Thus, at least we can say that the user perception is mostly the same. Note that we cannot

[6]The test statistic calculated by Shapiro-Wilk test [54], a method to identify whether given data follows the normal distribution, was 0.867 for our IOCs and 0.89 for the manually-generated ones.

Table 7: The requests to an IOC signature from the SOC analysts.

Requests	N
A description of the false positives.	3
Information on C&C servers.	2
A description pointer to external information sources.	2
A description of the intention of IOC.	1
A description of the artifact rarity.	1
A description of which stage's payload the IOC points to for multi-stage malware.	1
A description being self-contained so that analysts don't need to seek external sources.	1
Exact match between the content of the natural language description and the content covered by the IOC.	1

and do not aim to say that user perception is statistically equivalent just because there is no significant difference. Yet, there is no inferiority in user perception between automatically-generated and manually-generated IOCs. The results show that EIGER has enough capability to generate interpretable IOCs while achieving accuracy.

The results of the descriptive question in Table 7. While our IOCs have achieved enough interpretability, analysts had requested further features to the IOCs. Interestingly, most of them is about description. Whereas manually-generated those contain a description written in natural language, ours do not. Thus, through enhancing our method to provide it, we may generate more interpretable IOC even than manually-generated ones. We leave it as a future work.

9 DISCUSSION

While EIGER have reached our goals, it is necessary to discuss its limitation. Especially, we here assume an adversarial setting, *i.e.,* the situation when an attacker knows our method. In this section, we summarize possible attacks against our method and discuss their mitigations. Specifically, we organize the attack to our method into two types: evasion and denial-of-service (DoS).

Evasion. We here call an attack that avoids IOC-based detection as an evasion. The basic idea of evasion is to let malware create only artifacts that cannot be distinguished from other families or benign programs. For example, if only the Run key [38] is recorded in the log, the malware cannot be distinguished from other families by IOC. Similarly, malware that creates only an artifact that is identical to a benign program is difficult to classify. Furthermore, it is practically impossible to detect malware composed only of artifacts consisting of random character types and string lengths. This is because this type of malware cannot be distinguished from other malware families or benign programs at the IOC level. Unfortunately, this case was seen in the family called Ygdata[7] and Zlob[8] which creates a file with a fully-randomized name in a temporary directory and we could not observe any other artifacts creation. We face difficulty in detecting them while we rely on the concept of IOC.

In order to detect such malware, it is necessary to make use of more detailed features regardless of the IOC format. Though, it does not mean that our method becomes completely useless because this is a design issue of the IOC, it does not restrict our method itself. If artifacts such as disassembled code and memory layout that cannot be completely randomized are converted into the dataset format, EIGER can be applied as a signature generation method, even though the resulting products may deviate from the IOC category.

Denial of service. We discuss an attack surface that leads to a DoS at each step of EIGER. While this issue was not actualized in our experiment, it may become apparent in the adversarial setting.

For Candidate Enumeration step, we can assume an attack that makes it difficult to generate regular expression patterns by creating a large number of artifacts. That is, one can create a dedicated malware which exploits the algorithmic complexity of our clustering *i.e.,* $O(N^2 \log N)$. We may deal with this problem by using non-hierarchical clustering [16, 43] or by targeting only artifacts common to multiple samples.

For Optimization step, as our IOC generation builds on regular expression, there is a room for ReDoS attacks [13] against the regular expression engine. To this end, one can create a dedicated malware which leaves an artifact for ReDoS attack that delays the IOC coverage measurement. This issue can be addressed by providing a timeout for computing *incorrect-cover etc.* If the time taken to measure the coverage exceeds the required time, we can determine that this is an artifact created for ReDoS and can delete corresponding IOC the candidates.

10 RELATED WORK

Indicators of Compromise. IOCs [11, 12, 19] are common in sharing information on malware [15, 45] and thus become an integral part of today's security operation. Pioneer works [25, 34, 62] focused on extracting IOCs from human-written online report utilizing natural language processing techniques. Thus, on the automation of IOC generation for malware detection from non-human-written traces has not been studied yet.

Malware detection and classification. In this field, many behavioral features have been long-established *e.g.,* instruction semantics [9], API call sequences [8, 22, 31, 42, 51], call graph [29] and data flow [58]. Interested readers can refer to the comprehensive survey [61]. In contrast, our aim is to detect malware based on only coarse-grained features, that is, artifacts that can be obtained directly from most endpoints, rather than their fine-grained features.

Same as EIGER, Mastino [47] and MARMITE [56] use only a small piece of information obtainable from an endpoint, however, they simultaneously require network monitoring and thus they are not suitable for postmortem analysis of an endpoint alone. There

[7] MD5: `8903f0def1993bdbe1a05952766c21ee`
[8] MD5: `c0e5ec2f7841ba28557c2d49413f0365`

is also a case study [6] that detects suspicious programs using only endpoint-obtainable features, but it is not intended to provide an interpretable explanation. EIGER and they are complementary to each other.

Our idea of using regular expressions is inspired by the network signature generation methods [40, 44, 55], however, there are three differences between EIGER and them. First, whereas they are for a network, EIGER is for an endpoint. Second, they do not follow our *enumerate-then-optimize* design. In particular, they generate regular expressions with a single abstraction-level and select the generated expressions with given a threshold. In other words, they do not generate potentially-enumerable regular expressions and do not compute (near-)optimal solution, while EIGER does. Third, EIGER is designed to satisfy the domain-specific criteria *i.e.,* interpretability. In summary, EIGER can be regarded as their systematized variant for an endpoint.

Interpretability. Interpretability is an active area in a machine learning research. A model lacking this element could not be trusted by the user (analysts for us). We shall briefly discuss the two research direction to address this issue, instead of the comprehensive survey [23]. First is to explain the decision boundary of a black box model for a certain data point [36, 48]. In a malware domain, there are interesting works on explaining a Deep Neural Networks' decision to facilitate manual reverse engineering [24, 60]. However, they cannot be applicable to the IOC generation because they rely on fine-grained features and do only provide an explanation for a one data point at once. Second is to create a transparent model for the entire dataset that is easy to interpret. EIGER is positioned to this category since IDS framework [32, 33] lies in this category. The authors of the original IDS paper confirms through a user study that IDS generates rules that are more interpretable than rules generated by other machine learning techniques. However, since the original IDS is not designed to generate IOCs, the results have not been compared with manually-created IOCs. The difference between our work and them is that we extended IDS for malware detection purpose and compared the products with manually-generated IOCs.

11 CONCLUSION

We presented EIGER, a method to automatically generate accurate and interpretable IOCs from given malware traces. Our experiments on the accuracy showing a TPR of 91.98% and a FPR of 0.97% demonstrated that our IOCs are as accurate as other machine learning techniques. We also confirmed that EIGER can automatically refine the IOC and achieve high precision even under an adverse condition. Our user study with the 15 SOC analysts on the interpretability demonstrated that our IOCs bear comparison with manually-generated ones. While EIGER does not supersede existing malware detection method, but these experimental results indicate that EIGER is an appealing complement to endpoint malware detection in the real-world security operations.

The applicability of EIGER goes beyond the endpoint field. Our design, which explicitly introduces the steps of *enumerate-then-optimize*, enables us to optimize the combination of signature candidates with different abstraction-levels while guaranteeing the near-optimality. A promising avenue for the future in battle against malware threats is to apply our method to other fields which require the signature generation.

REFERENCES

[1] Rakesh Agrawal and Ramakrishnan Srikant. 1994. Fast Algorithms for Mining Association Rules in Large Databases. In *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB'94)*. 487–499.

[2] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, and Konrad Rieck. 2014. DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket. In *Proceedings of Network and Distribution System Security (NDSS'14)*.

[3] Peter Brass. 2008. *Advanced Data Structures* (1 ed.). Cambridge University Press, New York, NY, USA.

[4] Leo Breiman. 2001. Random Forests. *Mach. Learn.* 45, 1 (Oct. 2001), 5–32.

[5] Leo Breiman, Jerome Friedman, R. A. Olshen, and Charles J. Stone. 1984. *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA.

[6] Ahmet S. Buyukkayhan, Alina Oprea, Zhou Li, and William Robertson. 2017. Lens on the Endpoint: Hunting for Malicious Software Through Endpoint Data Analysis. In *Proceedings of the 20th International Symposium on Research in Attacks, Intrusions, and Defenses (RAID'17)*. 73–97.

[7] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'16)*. 785–794.

[8] Mihai Christodorescu, Somesh Jha, and Christopher Kruegel. 2007. Mining Specifications of Malicious Behavior. In *Proceedings of the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering (ESEC-FSE '07)*. 5–14.

[9] Mihai Christodorescu, Somesh Jha, Sanjit A. Seshia, Dawn Song, and Randal E. Bryant. 2005. Semantics-aware malware detection. In *2005 IEEE Symposium on Security and Privacy (S&P'05)*. 32–46.

[10] Fady Copty, Matan Danos, Orit Edelstein, Cindy Eisner, Dov Murik, and Benjamin Zeltser. 2018. Accurate Malware Detection by Extreme Abstraction. In *Proceedings of the 34th Annual Computer Security Applications Conference (ACSAC'18)*. 101–111.

[11] MITRE Corporation. [n. d.]. CybOX: Cyber Observable eXpression. http://cyboxproject.github.io.

[12] MITRE Corporation. [n. d.]. STIX: Structured Threat Information eXpression. http://stixproject.github.io.

[13] Scott A. Crosby and Dan S. Wallach. 2003. Denial of Service via Algorithmic Complexity Attacks. In *Proceedings of the 12th Conference on USENIX Security Symposium (USENIX Security'03)*. 3–3.

[14] CrowdStrike. [n. d.]. Hybrid Analysis. https://www.hybrid-analysis.com/.

[15] AT&T Cybersecurity. [n. d.]. AlienVault - Open Threat Exchange. https://otx.alienvault.com/.

[16] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 1996. A Density-based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD'96)*. 226–231.

[17] Feargus Pendlebury, Fabio Pierazzi, Roberto Jordaney, Johannes Kinder, and Lorenzo Cavallaro. 2019. TESSERACT: Eliminating Experimental Bias in Malware Classification across Space and Time. In *Proceedings of the 28th USENIX Security Symposium (USENIX Security'19)*.

[18] Uriel Feige, Vahab S. Mirrokni, and Jan Vondrák. 2011. Maximizing Non-monotone Submodular Functions. *SIAM J. Comput.* 40, 4 (July 2011), 1133–1153.

[19] FireEye. [n. d.]. OpenIOC. http://www.openioc.org.

[20] CIS Center for Internet Security. [n. d.]. Top 10 Malware January 2019. https://www.cisecurity.org/blog/top-10-malware-january-2019/.

[21] Cuckoo Foundation. [n. d.]. Cuckoo Sandbox. https://cuckoosandbox.org.

[22] Matt Fredrikson, Somesh Jha, Mihai Christodorescu, Reiner Sailer, and Xifeng Yan. 2010. Synthesizing Near-Optimal Malware Specifications from Suspicious Behaviors. In *Proceedings of the 2010 IEEE Symposium on Security and Privacy (S&P '10)*. 45–60.

[23] Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. 2018. A Survey of Methods for Explaining Black Box Models. *ACM Comput. Surv.* 51, 5, Article 93 (Aug. 2018), 42 pages.

[24] Wenbo Guo, Dongliang Mu, Jun Xu, Purui Su, Gang Wang, and Xinyu Xing. 2018. LEMNA: Explaining Deep Learning Based Security Applications. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS'18)*. 364–379.

[25] Ghaith Husari, Ehab Al-Shaer, Mohiuddin Ahmed, Bill Chu, and Xi Niu. 2017. TTPDrill: Automatic and Accurate Extraction of Threat Actions from Unstructured Text of CTI Sources. In *Proceedings of the 33rd Annual Computer Security Applications Conference (ACSAC'17)*. 103–115.

[26] Anil K. Jain, M. N. Murty, and P. J. Flynn. 1999. Data Clustering: A Review. *ACM Comput. Surv.* 31, 3 (Sept. 1999), 264–323.

[27] Yuhei Kawakoya, Makoto Iwamura, Eitaro Shioji, and Takeo Hariu. 2013. API Chaser: Anti-analysis Resistant Malware Analyzer. In *Proceedings of the 16th International Symposium on Research in Attacks, Intrusions, and Defenses (RAID'13)*.

123–143.
[28] Samir Khuller, Anna Moss, and Joseph (Seffi) Naor. 1999. The Budgeted Maximum Coverage Problem. *Inf. Process. Lett.* 70, 1 (April 1999), 39–45.
[29] Joris Kinable and Orestis Kostakis. 2011. Malware Classification Based on Call Graph Clustering. *J. Comput. Virol.* 7, 4 (Nov. 2011), 233–245.
[30] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *Proceedings of the 6th International Conference on Learning Representations (ICLR'15)*.
[31] Clemens Kolbitsch, Paolo Milani Comparetti, Christopher Kruegel, Engin Kirda, Xiaoyong Zhou, and XiaoFeng Wang. 2009. Effective and Efficient Malware Detection at the End Host. In *Proceedings of the 18th Conference on USENIX Security Symposium (USENIX Security'09)*. 351–366.
[32] Himabindu Lakkaraju, Stephen H. Bach, and Jure Leskovec. 2016. 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 (KDD'16)*. 1675–1684.
[33] Himabindu Lakkaraju, Stephen H. Bach, and Jure Leskovec. 2016. Interpretable Decision Sets: A Joint Framework for Description and Prediction. *Technical Report, Stanford InfoLab.*
[34] Xiaojing Liao, Kan Yuan, XiaoFeng Wang, Zhou Li, Luyi Xing, and Raheem Beyah. 2016. Acing the IOC Game: Toward Automatic Discovery and Analysis of Open-Source Cyber Threat Intelligence. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS'16)*. 755–766.
[35] Chronicle LLC. [n. d.]. VirusTotal - Free Online Virus, Malware and URL Scanner. https://www.virustotal.com/.
[36] Scott M. Lundberg and Su-In Lee. 2017. A Unified Approach to Interpreting Model Predictions. In *Advances in Neural Information Processing Systems 30*, 4765–4774.
[37] Mandiant. [n. d.]. IOC Editor User Guide (version 2.2.0.0). https://www.fireeye.com/content/dam/fireeye-www/services/freeware/ug-ioc-editor.pdf.
[38] Microsoft. [n. d.]. Run and RunOnce Registry Keys - Windows applications | Microsoft Docs. https://docs.microsoft.com/en-us/windows/desktop/setupapi/run-and-runonce-registry-keys.
[39] Vinod Nair and Geoffrey E. Hinton. 2010. Rectified Linear Units Improve Restricted Boltzmann Machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning (ICML'10)*. 807–814.
[40] Terry Nelms, Roberto Perdisci, and Mustaque Ahamad. 2013. ExecScent: Mining for New C&C Domains in Live Networks with Adaptive Control Protocol Templates. In *Presented as part of the 22nd USENIX Security Symposium (USENIX Security'13)*. 589–604.
[41] George L. Nemhauser, Laurence A. Wolsey, and Marshall L. Fisher. 1978. An Analysis of Approximations for Maximizing Submodular Set functions–I. *Math. Program.* 14, 1 (Dec. 1978), 265–294.
[42] Razvan Pascanu, Jack W. Stokes, Hermineh Sanossian, Mady Marinescu, and Anil Thomas. 2015. Malware classification with recurrent networks. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP'15)*. 1916–1920.
[43] Dan Pelleg and Andrew W. Moore. 2000. X-means: Extending K-means with Efficient Estimation of the Number of Clusters. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML'00)*. 727–734.
[44] Roberto Perdisci, Wenke Lee, and Nick Feamster. 2010. Behavioral Clustering of HTTP-based Malware and Signature Generation Using Malicious Network Traces. In *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation (NSDI'10)*. 26–26.
[45] MISP project. [n. d.]. MISP - Open Source Threat Intelligence Platform & Open Standards For Threat Information Sharing. http://www.misp-project.org/.
[46] YaraRules Project. [n. d.]. YaraRules. https://yararules.com/.
[47] Babak Rahbarinia, Marco Balduzzi, and Roberto Perdisci. 2016. Real-Time Detection of Malware Downloads via Large-Scale URL → File → Machine Graph Mining. In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security (ASIACCS '16)*. 783–794.
[48] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'16)*. 1135–1144.
[49] Konrad Rieck, Thorsten Holz, Carsten Willems, Patrick Düssel, and Pavel Laskov. 2008. Learning and Classification of Malware Behavior. In *Proceedings of the 5th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA'08)*. 108–125.
[50] Christian Rossow, Christian J. Dietrich, Chris Grier, Christian Kreibich, Vern Paxson, Norbert Pohlmann, Herbert Bos, and Maarten van Steen. 2012. Prudent Practices for Designing Malware Experiments: Status Quo and Outlook. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy (S&P'12)*. 65–79.
[51] Ashkan Sami, Babak Yadegari, Hossein Rahimi, Naser Peiravian, Sattar Hashemi, and Ali Hamze. 2010. Malware Detection Based on Mining API Calls. In *Proceedings of the 2010 ACM Symposium on Applied Computing (SAC'10)*. 1020–1025.
[52] Shlomo S. Sawilowsky and R. Clifford Blair. 1992. A more realistic look at the robustness and Type II error properties of the t test to departures from population normality. *Psychological Bulletin* 111, 2 (1992), 352–360.
[53] Marcos Sebastián, Richard Rivera, Platon Kotzias, and Juan Caballero. 2016. AVclass: A Tool for Massive Malware Labeling. In *Proceedings of the 19th International Symposium on Research in Attacks, Intrusions, and Defenses (RAID'16)*. 230–253.
[54] Samuel S. Shapiro and Martin B. Wilk. 1965. An analysis of variance test for normality (complete samples). *Biometrika* 52, 3-4 (12 1965), 591–611.
[55] Ben Stock, Benjamin Livshits, and Benjamin Zorn. 2016. Kizzle: A Signature Compiler for Detecting Exploit Kits. In *Proceedings of the 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'16)*. 455–466.
[56] Gianluca Stringhini, Yun Shen, Yufei Han, and Xiangliang Zhang. 2017. Marmite: Spreading Malicious File Reputation Through Download Graphs. In *Proceedings of the 33rd Annual Computer Security Applications Conference (ACSAC'17)*. 91–102.
[57] Frank Wilcoxon. 1945. Individual Comparisons by Ranking Methods. *Biometrics Bulletin* 1, 6 (1945), 80–83.
[58] Tobias Wüchner, Martín Ochoa, and Alexander Pretschner. 2015. Robust and Effective Malware Detection Through Quantitative Data Flow Graph Metrics. In *Proceedings of the 12th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA'15)*. 98–118.
[59] Khaled Yakdan, Sergej Dechand, Elmar Gerhards-Padilla, and Matthew Smith. 2016. Helping Johnny to Analyze Malware: A Usability-Optimized Decompiler and Malware Analysis User Study. In *Proceedings of the 2016 IEEE Symposium on Security and Privacy (S&P'16)*. 158–177.
[60] Hiromu Yakura, Shinnosuke Shinozaki, Reon Nishimura, Yoshihiro Oyama, and Jun Sakuma. 2018. Malware Analysis of Imaged Binary Samples by Convolutional Neural Network with Attention Mechanism. In *Proceedings of the 8th ACM Conference on Data and Application Security and Privacy (CODASPY'18)*. 127–134.
[61] Yanfang Ye, Tao Li, Donald Adjeroh, and S. Sitharama Iyengar. 2017. A Survey on Malware Detection Using Data Mining Techniques. *ACM Comput. Surv.* 50, 3, Article 41 (June 2017), 40 pages.
[62] Ziyun Zhu and Tudor Dumitras. 2018. ChainSmith: Automatically Learning the Semantics of Malicious Campaigns by Mining Threat Intelligence Reports. In *Proceedings of the 2018 IEEE European Symposium on Security and Privacy (EuroS&P'18)*. 458–472.