EIGER
Automated IOC Generation for Accurate and Interpretable Endpoint Malware Detection

NTT Secure Platform Laboratories; NTT Security (Japan) KK; Waseda University / NICT; University of California, Berkeley

Yuma Kurogome, Yuto Otsuki, Yuhei Kawakoya, Makoto Iwamura, Shogo Hayashi, Tatsuya Mori, and Koushik Sen
Malware Detection / Classification

- **State-of-the-art: Machine learning pipeline**
  - Works quite well, but requires rich information can be obtained at a sandbox

- **Endpoint**
  - API traces are not always available in all environments
  - Event logs are not always ...
  - What can we do with minimal endpoint-obtainable information?
IOC – Indicator of Compromise

• Behavioral signature used in the industry
  – Aims to complement network perimeter defenses
    • Antivirus
    • Sandbox w/ ML pipeline
  – Indicates creations of malicious *artifacts*:
    • File
    • Registry
    • Process
    • ...
  – Written in regex

Disclaimer: In this work, we did not consider the static IOC *i.e.*, YARA
IOC – Indicator of Compromise

• Used in a *human-in-the-loop* workflow
  – SOC Security Operation Center
    • EDR Endpoint Detection & Response
  – CSIRT Computer Security Incident Response Team
    • Forensic analysis
Automated IOC Generation – Challenges

• To imitate analysts’ decisions
  – How to extract malicious artifacts from analysis logs?
  – How to find out artifact which allows us to distinguish (Family A) and (Family B)?
  – How to accurately write a regex of them?

• While satisfying the following requirements:
  – **Accuracy**
    • An artifact with a high recall and precision should be expressed in an appropriate abstraction-level to classify a malware family
      – One artifact can be expressed as multiple regex patterns: IOC, [A-Z]{3}, .*, etc.
  – **Interpretability**
    • No inferiority in analysts’ perception between automatically-generated IOCs and manually-generated IOCs
      – Because IOCs are used in the human-in-the-loop
Motivating Example

Input:

\[\text{C:UsersJohnAppDataLocalTempaareg.exe} \]
\[\text{C:UsersJaneAppDataLocalTempinstall.vbs} \]

Existing regex-based methods

\[\text{C:Users.*AppDataLocalTemp.*} \]

*Interpretable, but inaccurate*
Too generic IOCs may be generated

Existing ML-based methods

\[\text{if get_weight(“AppData”) > 0.37 and if get_weight(“Local”) > 0.48 and if not get_weight(“Local”) > 0.6 and if get_weight(“Temp”) > 0.53 ..} \]

*Accurate, but uninterpretable*
Too complex IOCs may be generated

Desired output:

\[\text{C:Users.*AppDataLocalTemp(aareg.exe|install.vbs)} \]
EIGER – Exhaustive IOC Generator

• Our idea: **enumerate-then-optimize**
  – EIGER first enumerates candidates of an IOC with *multiple abstraction-levels*:
    • Input:
      – ACSAC, ACCEPT, ACCESS
    • Output:
  – ... then optimizes the combination
    • Submodular function maximization
      – Finds a subset of the IOC candidates which maximizes both accuracy and interpretability
      – This enables us to guarantee a near-optimal solution
    • How to formalize *interpretability*?
      – We borrowed the idea from the Interpretable Decision Set [KDD’16]
      – To explain a decision boundary as simple as possible
EIGER – Exhaustive IOC Generator

• A successor of the regex-based signature generation methods
  – Our design can be applicable to existing signature generation techniques
    • Pioneers generate signatures with a single level
    • Pioneers use a given threshold; there is no approx. guarantee
Overview

(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.

---

Enumerate

Optimize
**Enumeration**

- For each cluster, EIGER divides artifacts into each hierarchy
  - For example, file path is divided with ¥
- For each hierarchy, EIGER enumerates candidate patterns with three levels:
  - **Hard-coded pattern**
    - Input:
      - ACSAC, ACCEPT
    - Output
      - AC(SAC|CEPT)
    - Based on Trie structure
  - **Character class with range pattern**
    - Input:
      - ACCEPT, ACCESS
    - Output:
      - [A-Z]6
    - If there are the same length strings
  - **Arbitrary string pattern**
    - Input
      - ACSAC, ACCEPT
    - Output
      - .*
Optimization

$\text{argmax}_{R \subseteq S \times C} \sum_{i=1}^{6} \lambda_i f_i(R)$

$R = \{(s_1, c_1), \ldots, (s_k, c_k)\}$

To maximize the objective functions

Hyperparameters

IOC Candidates

IOCs
## Optimization

**IOC Candidates**

\[
\text{argmax}_{R \subseteq S \times C} \sum_{i=1}^{6} \lambda_i f_i(R)
\]

- \( R = \{(s_1, c_1), \ldots, (s_k, c_k)\} \)
- \( \lambda_i \): Weight of the \( i \)-th objective

**Hyperparameters**

To maximize the objective functions

**Objective Functions**

1. \( f_1(R) = |S| - \text{size}(R) \)
   - The number of IOCs
   - To represent the entire dataset with a few IOCs

2. \( f_2(R) = \sum_{r \in R} (\max_{r \in S \times C} \text{width}(r) - \text{width}(r)) \)
   - The number of regex in a rule
   - To favor simple IOCs

3. \( f_3(R) = \sum_{r_i, r_j \in R \atop i \leq j \atop c_i \neq c_j} (N - |\text{overlap}(r_i, r_j)|) \)
   - To reduce the intra-family overlap of IOCs

\( N \) is the total number of classes.

\( \text{overlap}(r_i, r_j) \) is the overlap between two classes.

\( \text{cover}(r_i) \) is the class covered by a rule. 

\( \text{width}(r) \) is the width of a rule.
Optimization

IOCs

Hyperparameters

To maximize the objective functions

\[
\text{argmax}_{R \subseteq S \times C} \sum_{i=1}^{6} \lambda_i f_i(R)
\]

**Objective Functions**

1. \(f_1(R) = |S| - \text{size}(R)\)
   - The number of IOCs
   - To represent the entire dataset with a few IOCs

2. \(f_2(R) = \sum_{r \in R} (\max_{r \in S \times C} \text{width}(r) - \text{width}(r))\)
   - The number of regex in a rule
   - To favor simple IOCs

3. \(f_3(R) = \sum_{r_i, r_j \in R, i \leq j, c_i \neq c_j} (N - |\text{overlap}(r_i, r_j)|) / \text{cover}(r_i) \cap \text{cover}(r_j)\)
   - To reduce the intra-family overlap of IOCs

4. \(f_4(R) = |C| - \sum_{c \in C} 1(\exists r = (s, c) \in R \text{ such that } c = c')\)
   - To have at least one IOC for each malware family

5. \(f_5(R) = N \cdot |S| - \sum_{r \in R} (\text{incorrect} - \text{cover}(r)) / \text{cover}(r) \setminus \text{correct} - \text{cover}(r)\)
   - To reduce false positives

6. \(f_6(R) = N - \sum_{(x, y) \in D} 1(|\{r | (x, y) \in \text{correct} - \text{cover}(r)\} | \geq 1) / |\{(x, y) \in \text{cover}(r) | y = c\}|\)
   - To have at least one accurate IOC for each data point

\(R = \{(s_1, c_1), \ldots, (s_k, c_k)\}\)

\(\text{regex class}\)
Experimental Results – Accuracy

- EIGER vs ML-based methods
  - Procedure
    - Generated IOCs / trained models from a train set
    - Tuned their parameters through 5-fold cross validation of a train set
    - Evaluated IOCs / models with a test set
EIGER vs ML-based methods

- Procedure
  - Created a dataset from samples
    - Collected from VirusTotal
    - Labeled by AVCLASS
    - Executed in 30 min on API Chaser sandbox to extract write events for file paths, registry key/values, process arguments – only endpoint-obtainable artifacts were extracted

## Experimental Results – Accuracy

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Label</th>
<th># classes</th>
<th># samples</th>
<th>Date range</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Train</strong></td>
<td>Malware</td>
<td>289</td>
<td>80,239</td>
<td>Nov 2018 – Feb 2019</td>
</tr>
<tr>
<td></td>
<td>Benign</td>
<td>1</td>
<td>9,664</td>
<td></td>
</tr>
<tr>
<td><strong>Test</strong></td>
<td>Malware</td>
<td>287</td>
<td>63,993</td>
<td>Mar 2019 – Apr 2019</td>
</tr>
<tr>
<td></td>
<td>Benign</td>
<td>1</td>
<td>8,130</td>
<td></td>
</tr>
</tbody>
</table>
Experimental Results – Accuracy

- EIGER vs ML-based methods
  - Result
    - Our IOCs successfully detected malware as accurate as known ML-based methods

<table>
<thead>
<tr>
<th>Method</th>
<th>FPR%</th>
<th>TPR%</th>
<th>Precision%</th>
<th>Recall%</th>
</tr>
</thead>
<tbody>
<tr>
<td>EIGER</td>
<td>0.97</td>
<td>91.18</td>
<td>99.87</td>
<td>95.76</td>
</tr>
<tr>
<td>Random Forest</td>
<td>1.19</td>
<td>93.59</td>
<td>99.84</td>
<td>96.61</td>
</tr>
<tr>
<td>XGBoost</td>
<td>2.20</td>
<td>91.31</td>
<td>99.69</td>
<td>95.32</td>
</tr>
<tr>
<td>Neural Network</td>
<td>1.00</td>
<td>93.82</td>
<td>99.87</td>
<td>96.75</td>
</tr>
</tbody>
</table>
Experimental Results – Accuracy

- **EIGER vs ML-based methods**
  - **Result**
    - Why EIGER showed a high precision?
      - If malware can be detected from the presence or absence of a single artifact, there is no need to consider a benign weights
      - Even if the malware creates a benign artifact, the detection criteria of an IOC will not be loosened

\[
\text{if malicious then malware}
\]

\[
\text{if get_weight("malicious") > 0.37 and if get_weight("benign") < 0.48 then malware}
\]
Experimental Results – Accuracy

• EIGER vs ML-based methods
  – Caveats
    • The performance of ML-based methods would be further improved by including rich features such as API trace
    • The focus here was on a comparison based on the coarse-grained, endpoint-obtainable features
    • Our aim was not beating ML-based methods
Experimental Results – Accuracy

• EIGER vs Real-world EDR product
  – Procedure
    • Deployed a certain EDR product A
      – Has built-in IOCs and an interface of IOCs addition
    • Put EIGER IOCs, generated at the previous experimentation, into the product
    • Run samples and compared detections between our IOCs and the built-in IOCs
  • Samples
    – Collected from Hybrid-Analysis, no-overlap of the source of EIGER IOCs
    – Binary-labeled (malicious / benign) based on Hybrid-Analysis’s score
    – We here conducted a binary classification due to the product’s specs

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Label</th>
<th># samples</th>
<th>Date range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test</td>
<td>Malware</td>
<td>939</td>
<td>Apr 2019</td>
</tr>
<tr>
<td></td>
<td>Benign</td>
<td>808</td>
<td></td>
</tr>
</tbody>
</table>
Experimental Results – Accuracy

- **EIGER vs Real-world EDR product**

  - **Result**
    - Our IOCs could only detect 133 samples out of 939 malware samples
    - Negative result? – wait!
    - Due to the dataset difference
    - Yet, EIGER IOCs showed 98.58 Precision%  

- EIGER covered malware that the real-world product could not detect

---

![Venn Diagram](image)

<table>
<thead>
<tr>
<th>Product A (77)</th>
<th>EIGER (133)</th>
</tr>
</thead>
<tbody>
<tr>
<td>40 Intersections</td>
<td>133</td>
</tr>
</tbody>
</table>
Experimental Results – Accuracy

• EIGER vs Real-world EDR product
  – Caveats
    • The built-in IOCs focus on generic malware behavior
    • EIGER IOCs focus on family-specific behavior
    • The latter can change with the advent of new malware, thus continuous IOC generation is required in a real-world security operation
Experimental Results – Interpretability

• EIGER vs Manually-generated IOCs
  – Procedure
    • Recruited 15 SOC analysts working for NTT Security Japan
      – Provide outsourced monitoring and management of security devices and systems
      – Use IOCs on a daily basis
    • Offered them to provide our IOCs after the experiment
      – To motivate them to join the experiment
    • Presented randomly-chosen 6 EIGER IOCs / manually-generated IOCs
      – While hiding creator information (EIGER or the analyst)
      – Through our online survey platform
    • Created 8 questions on interpretability
      – Based on a usability-aware decompiler study [S&P’16]
      – Conducted a pre-study with the creator of the manually-generated IOCs
      – Refined the sentences of the questions
Experimental Results – Interpretability

• EIGER vs Manually-generated IOCs
  – Procedure
    • Asked the questions
      – When we got a question from an analyst, we dealt with it on the moment
    • Collected 5-point Likert scale answers
      – Strongly agree, agree, neutral, disagree, strongly disagree
    • Performed Wilcoxon signed-rank test on the answers
      – Existing IOCs are created by analysts and are hence *interpretable* to analysts
      – If there is no significant difference in user reaction, we can say EIGER IOCs are interpretable
Experimental Results – Interpretability

• EIGER vs Manually-generated IOCs
  – Result
    • The p-value was 0.55
      – Indicates no significant difference in user perceptions between EIGER IOCs and manually-generated IOCs
Experimental Results – Interpretability

• EIGER vs Manually-generated IOCs
  – Caveats
    • Representativeness
      – We cannot reach out the entire SOCs in the world; this experiment is limited in scope
    • Experimental design
      – There may be a question that makes a difference in analysts’ reaction
      – We focused on following the existing design [S&P’16] at this time
Experimental Results – Summary

• Accuracy
  – EIGER is comparable to the ML-based methods
  – EIGER is an appealing complement to commercial IOCs

• Interpretability
  – EIGER is able to generate IOCs as interpretable as humans
Limitations

• Evasion
  – Randomized artifact names; IOCs are mortal
  – Sandbox evasion is out-of-scope
Limitations

• Denial-of-Service
  – For the enumeration step:
    • Algorithmic complexity attack for the clustering
  – For the optimization step:
    • ReDoS
• IOC description
  – Analysts want a description of IOC written in natural language
Conclusion

• Automated IOC generation is feasible
  – EIGER produces interpretable, yet accurate IOCs
  – EIGER IOCs can be seamlessly incorporated into real-world security operations

• Enumerate-then-optimize design goes beyond the endpoint field
  – We can apply this principle to other signature generation techniques