Practical Adversarial Malware Example Attacks and Defenses

Fangtian Zhong
Gianforte School of Computing
Email: fangtian.zhong@montana.edu

2024.05.28
Who Am I?

Fangtian Zhong

--------Assistant professor
Montana State University

Research interests
- Software security
- Program analysis
- Machine learning for cybersecurity

Education
- Ph.D. in computer science, George Washington University
- B.E. in Software Engineering, Northeast Normal University

Postdoc Training
- University of Notre Dame
- Pennsylvania State University
Outline

1. Introduction
2. Adversarial Malware Example Attacks
3. Adversarial Malware Example Defenses
Section One

Introduction
Who needs to worry about software security?

Researchers

Software engineers

EVERYONE

- Bank records
- Medical records
- Credit card information
- Technical project data
- Etc.

Results of unsecure software
- Identity theft
- Confidential information leakage
- Loss of data
- Slow computer
- Denied service
- Financial loss...
Types of software security issues

- Malware attack
- Denial of service attack
- Software vulnerability exploitation
- Linearization attack
- Etc.
Section Two

Adversarial Malware Example Attacks MalFox
Adversarial Examples

Image Recognition

Original Image → ML system → Adversarial Example → ML system

Perturbation

Deceiving Attack

Hippo

Speech Recognition

Original Audio → ML system

Noise X 0.001 → Adversarial Example → ML system

“Software Security”

“Program Analysis”
Motivation

Malware

Perturbation Module

Adversarial Malware Example

Malicious

Malicious

Malicious

Malicious

Benign

Benign

Benign

Benign

Benign

Benign

Benign

Benign

Benign
Challenges-Adversarial Malware Example Generation

To generate practical adversarial malware examples

1. Instrumenting malware samples to attach perturbation.
2. Remaining the functionality of original malware samples.
3. Evading detection by malware detectors under a black-box setting.
MalFox Framework

PE Parser retrieves the dynamic link libraries (DLLs) and system functions in programs as features.

Generator produces a perturbation path that improves malware's evasive capability while maximizing the probability of Discriminator making a mistake.

PE Editor follows the perturbation path output from Generator to produce an adversarial malware example.

Discriminator estimates the probability of maliciousness for a malware program, and provides gradient information to train Generator.

Detector ensures the reliability of our datasets, provides labels for Discriminator, and validates the performance of MalFox.
## Algorithm 1 MalFox Training Procedure

1. Convert each malware and benignware program in the training dataset into a binary feature vector by PE Parser;

2. **while** not converging **do**

3. Sample a minibatch of malware feature vectors and three-dimensional Gaussian noises, combine each malware feature vector with a noise sample, and input the results to Generator;

4. Generator generates perturbation paths and inputs them to PE Editor;

5. PE Editor produces adversarial malware examples following the perturbation paths;

6. Sample a minibatch of benignware feature vectors;

7. Update Discriminator’s parameters with the adversarial malware examples and benignware programs by descending along the gradient of $L_D$;

8. Sample three-dimensional Gaussian noises, combine each with a malware feature vector in the minibatch, and input the results to Generator;

9. Generator generates perturbation paths and inputs them to PE Editor;

10. PE Editor produces adversarial malware examples following the perturbation paths;

11. Detector labels the adversarial malware examples;

12. Update Generator’s parameters with the newly generated adversarial malware examples by descending along the gradient of $L_G$;

13. **end while**

![Diagram](attachment:diagram.png)
MalFox Training and Test Procedure

How to generate a powerful adversarial malware example?

Start

PE Parser

Feature Extraction & Transformation

Binary Malware Feature Vector

Generator

Perturbation Path

PE Editor

Adversarial Malware Example

Malware

Gaussian Noise
typedef struct _IMAGE_IMPORT_DESCRIPTOR {
    union {
        DWORD Characteristics;
        DWORD OriginalFirstThunk;
    } DUMMYUNIONNN;
    DWORD TimeDataReferences;
    DWORD Forwarder.chain;
    DWORD Name;
    DWORD FirstThunk;
} IMAGE_IMPORT_DESCRIPTOR;

typedef struct _IMAGE_IMPORT_BY_NAME {
    WORD Hint;
    CHAR Name[1];
} IMAGE_IMPORT_BY_NAME, *PIMAGE_IMPORT_BY_NAME;
MalFox Component: PE Editor (Obfusmal)

Obfusmal

<table>
<thead>
<tr>
<th>malware.exe</th>
<th>Shell.dll</th>
</tr>
</thead>
<tbody>
<tr>
<td>code section</td>
<td></td>
</tr>
</tbody>
</table>

Adversarial malware example

1. Read malware.exe, obtain the address and size of its code section, and encrypt the code section;
2. Develop Shell.dll with the functionality that can store the crucial information, decrypt the code section, and jump to the start address of program execution (OEP) of malware.exe to execute the code;
3. Add a section with the length up to Shell.dll in malware.exe, and save Shell.dll in the newly added section of malware.exe.

OEP

Shell.dll

Phase I

Phase II

Phase III

Decrypt

Execute

01

02

03

malware.exe
MalFox Component: PE Editor (Stealmal)

<table>
<thead>
<tr>
<th>Stealmal</th>
<th>Shell.exe</th>
<th>malware.exe</th>
</tr>
</thead>
</table>

Adversarial malware example

2

1

Encrypt the entire malware.exe.

Develop a program Shell.exe with the functionality that can decrypt the malware, create a suspended process, obtain the process space, copy the malware into the space, change the context of the process to the entry point of the malware, and resume the process. And add a section in Shell.exe to save malware.exe.
## MalFox Component: PE Editor (Hollowmal)

<table>
<thead>
<tr>
<th>Hollowmal</th>
</tr>
</thead>
<tbody>
<tr>
<td>benignware.exe</td>
</tr>
</tbody>
</table>

**Adversarial malware example**

### 01
Select benignware, encrypt the entire malware.exe, and add a section in the benignware to save the encrypted malware.exe;

### 02
Develop a DLL named Hollow.dll embracing similar functionality as Shell.exe, and add another section in the benignware to save Hollow.dll following the encrypted malware.exe.
**MalFox Component: PE Editor (Combination)**

<table>
<thead>
<tr>
<th>Combination</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shell.exe</td>
</tr>
<tr>
<td>malware.exe</td>
</tr>
<tr>
<td>code section</td>
</tr>
<tr>
<td>Shell.dll</td>
</tr>
</tbody>
</table>

**Adversarial malware example**

**Diagram:**
- **Shell.exe**
  - **Phase I:** OEP
  - **Phase II:** Shell.exe
  - **Phase III:** Intermediate\textsubscript{adv}
  - **Phase IV:** Shell.dll
  - **Phase V:** code section

**Flow:**
- **Execute** from **Shell.exe** to **Phase II**
- **Execute** from **Phase II** to **Intermediate\textsubscript{adv}**
- **Decrypted** from **Intermediate\textsubscript{adv}** to **Phase III**
- **Execute** from **Phase III** to **Phase IV**
- **Decrypted** from **Phase IV** to **Phase V**
MalFox Component: Detector

VIRUSTOTAL

https://www.virustotal.com/gui/home/upload

1. Contains many antivirus products and online scan engines to check for malware.

2. Popular tools such as McAfee, F-Secure, Tencent, 360, and Microsoft in VirusTotal, have been widely adopted on laptop and mobile devices.

3. It is well received by security professionals and researchers.
MalFox Component: Discriminator
Accumulation results—Discriminator as the attack target

<table>
<thead>
<tr>
<th>Types</th>
<th>Training Dataset</th>
<th>Test Dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td>RF</td>
<td>97.62</td>
<td>95.38</td>
</tr>
<tr>
<td>LR</td>
<td>92.2</td>
<td>92.27</td>
</tr>
<tr>
<td>DT</td>
<td><strong>97.89</strong></td>
<td>93.98</td>
</tr>
<tr>
<td>SVM</td>
<td>93.11</td>
<td>93.13</td>
</tr>
<tr>
<td>MLP</td>
<td>95.11</td>
<td>94.89</td>
</tr>
<tr>
<td>VOTE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LSTM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BiLSTM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LSTM-Average</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BiLSTM-Average</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LSTM-Attention</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BiLSTM-Attention</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Accuracy is the ratio of incorrectly predicted adversarial malware examples \(a\), and \(A\) is all adversarial malware examples \(A\) by Discriminator.

\[
\text{accuracy} = \frac{a}{A}
\]
# Experiment Results - VirusTotal as The Attack Target

## Comparison Results (%)

<table>
<thead>
<tr>
<th>Evaluation Metrics</th>
<th>Average</th>
<th>Max</th>
<th>Min</th>
</tr>
</thead>
<tbody>
<tr>
<td>Detection Rate (Malware)</td>
<td>68.8</td>
<td>85.4</td>
<td>26.8</td>
</tr>
<tr>
<td>Detection Rate (Foxy Malware)</td>
<td>29.7</td>
<td>43.9</td>
<td>18.3</td>
</tr>
<tr>
<td>Evasive Rate (Foxy Malware)</td>
<td>56.2</td>
<td>74.6</td>
<td>9.1</td>
</tr>
</tbody>
</table>

**Detection rate** is calculated as:

\[ \text{detection rate} = \frac{n}{N} \]

Where:
- \( n \) is the number of entities that detect the malware,
- \( N \) is the total number of entities tested.

**Evasive rate** is calculated as:

\[ \text{evasive rate} = \frac{N_{\text{orig}} - N_{\text{adv}}}{N_{\text{orig}}} \]

Where:
- \( N_{\text{orig}} \) is the number of entities that detect the original malware,
- \( N_{\text{adv}} \) is the number of entities that detect the adversarial malware example.
Adversarial Malware Example Defenses
Weakness

After the generation of adversarial malware examples, existing malware detectors based on classification, either static-based or dynamic-based, should be improved.

Weaknesses

- High knowledge barriers for security engineers
- Complicated performance examination
- Infeasible reverse analysis for encrypted or compressed malware

Dynamic-based classification

- Huge computing burdens for computers
- Poor reliability due to specific inputs
- Complicated performance examination

Goal

Provide an efficient but simple classifier to distinguish different types of adversarial malware examples as well as other types of malware samples.
Challenges-The Efficient and Simple Defense

To provide an efficient, simple, and effective defense strategy

1. The processing time for classification being withstood by users.
2. Correctly distinguishing different malware families although encrypted or compressed.
The Overview of The Framework-VisMal

1. Convert to images
2. Apply contrast limited adaptive histogram equalization to images
3. Resize images to 64x64
4. Apply CNN to resized images for classification

Malware samples

Imaged data

Classifier

Feature Engineer

Predicted malware family

Adiaser.C Allaple.A Agent.FYI
C2LOP.P Allaple.L Fakerean
## VisMal Component: Convertor

### Correspondence between the malware sample file size and the converted image width

<table>
<thead>
<tr>
<th>File Size</th>
<th>Width</th>
<th>Height</th>
</tr>
</thead>
<tbody>
<tr>
<td>≤10KB</td>
<td>32</td>
<td>(0, 312]</td>
</tr>
<tr>
<td>10KB-30KB</td>
<td>64</td>
<td>(156, 468]</td>
</tr>
<tr>
<td>30KB-60KB</td>
<td>128</td>
<td>(234, 468]</td>
</tr>
<tr>
<td>60KB-100KB</td>
<td>256</td>
<td>(234, 390]</td>
</tr>
<tr>
<td>100KB-200KB</td>
<td>384</td>
<td>(260, 520]</td>
</tr>
<tr>
<td>200KB-500KB</td>
<td>512</td>
<td>(390, 976]</td>
</tr>
<tr>
<td>500KB-1000KB</td>
<td>768</td>
<td>(651, 1302]</td>
</tr>
<tr>
<td>≥1000KB</td>
<td>1024</td>
<td>(976, ∞)</td>
</tr>
</tbody>
</table>

1. Sequentially reads the binary data in bytes and converts each byte into a number ranging in [0-255]

2. Reshape the image data by following a recommended fixed width with a variable height
VisMal Component: Feature Engineer

\[ cdf(i) = \sum_{j=0}^{i} n_j, 0 \leq i < L \]  \hspace{1cm} (1)

\[ y = h_R \]  \hspace{1cm} (2)

where \( L \) is the total number of gray levels (typically 256), and \( n_j \) is the total of the cumulative distribution function calculated in in Eq (1), while \( cdf_{\text{max}} \) gives the maximum value.

\[ y_1 = \]  \hspace{1cm} (4)

\[ y_2 = \frac{x_{22} - x_2}{x_{22} - x_{12}} h_{R_{Q12}}(x_{12}) + \frac{x_2 - x_{12}}{x_{22} - x_{12}} h_{R_{Q22}}(x_{22}) \]  \hspace{1cm} (5)

\[ y' = \frac{y_2 - h_{R_p}(x')}{y_2 - y_1} y_1 + \frac{h_{R_p}(x') - y_1}{y_2 - y_1} y_2 \]  \hspace{1cm} (6)
Evaluation Results (Accuracy)

<table>
<thead>
<tr>
<th>Classification Method</th>
<th>Precision</th>
<th>Recall</th>
<th>F1 Score</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>LR</td>
<td>60.6</td>
<td>96.4</td>
<td>74.4</td>
<td>67.4</td>
</tr>
<tr>
<td>NB</td>
<td>76.3</td>
<td>11.2</td>
<td>19.5</td>
<td>54.6</td>
</tr>
<tr>
<td>KNN</td>
<td>81.2</td>
<td>81.2</td>
<td>81.2</td>
<td>81.5</td>
</tr>
<tr>
<td>DT</td>
<td>85.9</td>
<td>85.6</td>
<td>85.7</td>
<td>86.0</td>
</tr>
<tr>
<td>AB</td>
<td>67.2</td>
<td>89.5</td>
<td>76.7</td>
<td>73.3</td>
</tr>
<tr>
<td>RF</td>
<td>89.9</td>
<td>88.6</td>
<td>89.2</td>
<td>89.5</td>
</tr>
<tr>
<td>SVM</td>
<td>70.0</td>
<td>84.4</td>
<td>76.5</td>
<td>74.5</td>
</tr>
<tr>
<td>DNN</td>
<td>90.6</td>
<td>91.1</td>
<td>90.9</td>
<td>91.0</td>
</tr>
<tr>
<td>DRBA+CNN</td>
<td>94.6</td>
<td>94.5</td>
<td>94.5</td>
<td>94.5</td>
</tr>
<tr>
<td>VisMal without transformation</td>
<td>92.0</td>
<td>93.3</td>
<td>92.2</td>
<td>93.3</td>
</tr>
<tr>
<td>VisMal</td>
<td>95.3</td>
<td>96.0</td>
<td>95.2</td>
<td>96.0</td>
</tr>
</tbody>
</table>
## Evaluation Results (Efficiency)

### Comparison of The Classification Method

<table>
<thead>
<tr>
<th>Method</th>
<th>Extraction Time</th>
<th>Classification Time</th>
<th>Total Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nataraj et al. [34]</td>
<td>32.7 ms</td>
<td>2.1 ms</td>
<td>34.8 s</td>
</tr>
<tr>
<td>Cui et al. [44]</td>
<td>-</td>
<td>-</td>
<td>20 ms</td>
</tr>
<tr>
<td>Naeem et al. [45]</td>
<td>-</td>
<td>4.27s</td>
<td>-</td>
</tr>
<tr>
<td>Yuan et al. [36]</td>
<td>144.3 ms</td>
<td>191.5 ms</td>
<td>335.8 ms</td>
</tr>
<tr>
<td>Vasan et al. [46]</td>
<td>-</td>
<td>-</td>
<td>1.18 s</td>
</tr>
<tr>
<td>Verma et al. [35]</td>
<td>37 ms</td>
<td>10 ms</td>
<td>47 ms</td>
</tr>
<tr>
<td><strong>VisMal</strong></td>
<td><strong>0.3 ms</strong></td>
<td><strong>3.7 ms</strong></td>
<td><strong>4.0 ms</strong></td>
</tr>
</tbody>
</table>

*IEEE Transactions on Computers, pp. 1-14, 2022.*
Thanks for your listening!

Fangtian Zhong
Gianforte School of Computing
Email: fangtian.zhong@montana.edu

2024.05.28