Malware Makeover: Breaking ML-based Static Analysis by Modifying Executable Bytes

Keane Lucas, Mahmood Sharif, Lujo Bauer, Michael K. Reiter, Saurabh Shintre
Malware detection is fundamental for cybersecurity

Anti-virus software routinely needs to examine programs for potential threats
Malware detection is fundamental for cybersecurity

Anti-virus software routinely needs to examine programs for potential threats

Machine learning (ML) models show promise / are in use for detection
Malware detection is fundamental for cybersecurity

Anti-virus software routinely needs to examine programs for potential threats

Machine learning (ML) models show promise / are in use for detection

But, malware classification models may be susceptible to evasion
Malware detection is fundamental for cybersecurity

Anti-virus software routinely needs to examine programs for potential threats

Machine learning (ML) models show promise / are in use for detection

But, malware classification models may be susceptible to evasion

Creating useful defenses requires knowledge of how ML models can be attacked
Deep Neural Networks (DNNs) for Static Malware Detection

Program binary represented as variable length sequence of integers/bytes
- A single byte’s meaning depends on the values of bytes around it
- Byte values are treated as categorical
  - Absolute difference between byte values has no meaning

Deep Neural Networks (DNNs) for Static Malware Detection

Program binary represented as variable length sequence of integers/bytes
- A single byte’s meaning depends on the values of bytes around it
- Byte values are treated as categorical
  - Absolute difference between byte values has no meaning

Attacking ML Algorithms – Adversarial Examples

Attacks use classifier’s trained weights to craft imperceptible adversarial noise (or perturbations) to cause misclassification

- Fast Gradient Sign Method (FGSM)
- Projected Gradient Descent (PGD)

Attacking DNNs for Static Malware Detection

Must ensure all byte changes preserve binary functionality
Assume whitebox access to target model (can view trained weights)
• Our paper also examines a blackbox threat model

Creating Adversarial Examples from Binaries

To modify binaries without changing functionality, use functionality preserving transformations:

Creating Adversarial Examples from Binaries

To modify binaries without changing functionality, use functionality preserving transformations:

- **In-Place Replacement (IPR)**
  - Four types: preserv, swap, reorder, equiv

Reorder (1/4 IPR)

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>mov edx, [ebp+4]</td>
<td>8b5504</td>
</tr>
<tr>
<td>sub edx, -0x10</td>
<td>83eaf0</td>
</tr>
<tr>
<td>mov ebx, [ebp+8]</td>
<td>8b5d08</td>
</tr>
<tr>
<td>mov [ebx], edx</td>
<td>8913</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>mov ebx, [ebp+8]</td>
<td>8b5d08</td>
</tr>
<tr>
<td>mov edx, [ebp+4]</td>
<td>8b5504</td>
</tr>
<tr>
<td>sub edx, -0x10</td>
<td>83eaf0</td>
</tr>
<tr>
<td>mov [ebx], edx</td>
<td>8913</td>
</tr>
</tbody>
</table>


Creating Adversarial Examples from Binaries

To modify binaries without changing functionality, use functionality preserving transformations:

- **In-Place-Replacement (IPR)**
- Four types: preserv, swap, reorder, equiv
- **Displacement (Disp)**

**Attack Algorithm**

1. Random initialization

---

<table>
<thead>
<tr>
<th>Algorithm 1: White-box attack.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input</strong>: $\mathcal{F} = H(\mathcal{B}(\cdot)), L_{\mathcal{F}}, x, y, n_{\text{iters}}$</td>
</tr>
<tr>
<td><strong>Output</strong>: $\hat{x}$</td>
</tr>
</tbody>
</table>

1. $i \leftarrow 0$;
2. $\hat{x} \leftarrow \text{RandomizeAll}(x)$;
Attack Algorithm

1. Random initialization

2. For every function:
   a. Randomly choose from valid transformations
Attack Algorithm

1. Random initialization

2. For every function:
   a. Randomly choose from valid transformations
   b. Generate byte changes using chosen transformation and check gradient in embedding
Guided Transformations

1. Random initialization

2. For every function:
   a. Randomly choose from valid transformations
   b. Generate byte changes using chosen transformation
   c. If byte changes align with loss gradient – accept and move on to next part of function. If not, discard and go back to step b
   d. Execute until all instructions in function have been reached
Attack Algorithm

1. Random initialization

2. For every function:
   a. -- d. ...

3. Repeat step 2 until success or 200 iterations

Algorithm 1: White-box attack.

```plaintext
Input: \( \mathcal{F} = \mathbb{H}(\mathcal{B}(\cdot)), L, x, y, \text{niterations} \)
Output: \( \hat{x} \)

1. \( i \leftarrow 0; \)
2. \( \hat{x} \leftarrow \text{RandomizeAll}(x); \)
3. while \( \mathcal{F}(\hat{x}) = y \) and \( i < \text{niterations} \) do
   for \( f \in \hat{x} \) do
     \( \hat{e} \leftarrow \mathcal{E}(\hat{x}); \)
     \( g \leftarrow \mathcal{A}_{\mathcal{F}}(\hat{x}, y); \)
     \( o \leftarrow \text{RandomTransformationType}(); \)
     \( \hat{x} \leftarrow \text{RandomizeFunction}(\hat{x}, f, o); \)
     \( \hat{e} \leftarrow \mathcal{E}(\hat{x}); \)
     \( \delta_f = \hat{e}_f - \hat{e}_f; \)
     if \( g_f \cdot \delta_f > 0 \) then
       \( \hat{x} \leftarrow \hat{x}; \)
     end
   end
   \( i \leftarrow i + 1; \)
end
return \( \hat{x}; \)
```
Experiment Setup – Dataset

• 32-bit portable executable (PE) files, smaller than 5 MB, first seen in 2020, collected from VirusTotal feed (VTFeed), either 0 or >40 AV detections

<table>
<thead>
<tr>
<th>VTFeed</th>
<th>Train</th>
<th>Val.</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Benign</td>
<td>111,258</td>
<td>13,961</td>
<td>13,926</td>
</tr>
<tr>
<td>Malicious</td>
<td>111,395</td>
<td>13,870</td>
<td>13,906</td>
</tr>
</tbody>
</table>
Experiment Setup – Dataset

• 32-bit portable executable (PE) files, smaller than 5 MB, first seen in 2020, collected from VirusTotal feed (VTFeed), either 0 or >40 AV detections

• Labeled as benign (resp. malicious) if classified malicious by 0 (resp. >40) antivirus vendors aggregated by VirusTotal

<table>
<thead>
<tr>
<th>VTFeed</th>
<th>Train</th>
<th>Val.</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Benign</td>
<td>111,258</td>
<td>13,961</td>
<td>13,926</td>
</tr>
<tr>
<td>Malicious</td>
<td>111,395</td>
<td>13,870</td>
<td>13,906</td>
</tr>
</tbody>
</table>
Experiment Setup – Dataset

• 32-bit portable executable (PE) files, smaller than 5 MB, first seen in 2020, collected from VirusTotal feed (VTFeed), either 0 or >40 AV detections

• Labeled as benign (resp. malicious) if classified malicious by 0 (resp. >40) antivirus vendors aggregated by VirusTotal

• 139K benign and 139K malicious, shuffled, and randomly partitioned into Train (80%), Validation (10%), and Test (10%) sets
Experiment Setup – DNNs

State-of-the-art architectures we trained:

• MalConv – proposed by Raff et al.
• Avast – proposed by Krčál et al.

Endgame – pre-trained DNN (Anderson et al.)

• Based on MalConv architecture
• Trained on 600K binaries, evenly distributed between benign and malicious
• 92% detection rate when restricted to a false positive rate of 0.1%

Architecture diagram of MalConv model (from Raff et al.)

<table>
<thead>
<tr>
<th></th>
<th>Accuracy</th>
<th>TPR @ 0.1% FPR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Train</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Val.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AvastNet</td>
<td>99.89%</td>
<td>98.59%</td>
</tr>
<tr>
<td>MalConv</td>
<td>99.97%</td>
<td>98.67%</td>
</tr>
</tbody>
</table>
Results – DNNs and Malware Samples

Malware samples used to construct adversarial examples

- 100 sampled from VirusTotal (aggregates binaries and anti-virus vendor detections)
  - Unpacked
  - Size below models’ smallest input (512KB)
  - At least 40 anti-virus detections for malware
Experiment Setup – Measuring Success

Experiment methods

• 10 repetitions of each experiment

• Deemed successful if an attack can reduce maliciousness score to below 0.1% FPR threshold (0.5 for Endgame)
Experiment Setup – Measuring Success

Experiment methods

• 10 repetitions of each experiment

• Deemed successful if an attack can reduce maliciousness score to below 0.1% FPR threshold (0.5 for Endgame)

Two measures of success

• Coverage – fraction of binaries an attack was successful in at least one of the trials

Coverage = 3/5 = 60%
Experiment Setup – Measuring Success

Experiment methods

- 10 repetitions of each experiment
- Deemed successful if an attack can reduce maliciousness score to below 0.1% FPR threshold (0.5 for Endgame)

Two measures of success

- Coverage – fraction of *binaries* an attack was successful in *at least* one of the trials
- Potency – fraction of *trials* that succeeded, over all binaries

Coverage = \( \frac{3}{5} = 60\% \)

Potency = \( \frac{8}{25} = 32\% \)
Experiment Setup – Measuring Success

Experiment methods

- 10 repetitions of each experiment
- Deemed successful if an attack can reduce maliciousness score to below 0.1% FPR threshold (0.5 for Endgame)

Two measures of success

- Coverage – fraction of *binaries* an attack was successful in *at least* one of the trials
- Potency – fraction of *trials* that succeeded, over all binaries

<table>
<thead>
<tr>
<th>Trials</th>
<th>Binaries</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Coverage = 3/5 = 60%
Potency = 8/25 = 32%
Coverage ≥ Potency
Results – Overall

Attack success rates in the white-box setting
• Potency shown as lighter bars and coverage as darker bars
Results – Overall

Attack success rates in the white-box setting
• Potency shown as lighter bars and coverage as darker bars

Random < IPR
Results – Overall

Attack success rates in the white-box setting
• Potency shown as lighter bars and coverage as darker bars

Random < IPR
Results – Overall

Attack success rates in the white-box setting
• Potency shown as lighter bars and coverage as darker bars

Random < IPR
Results – Overall

Attack success rates in the white-box setting
  • Potency shown as lighter bars and coverage as darker bars

Random < IPR
Attack success rates in the white-box setting
- Potency shown as lighter bars and coverage as darker bars

Random < IPR < Disp
Results – Overall

Attack success rates in the white-box setting
• Potency shown as lighter bars and coverage as darker bars

Random < IPR < Disp
Results – Overall

Attack success rates in the white-box setting
• Potency shown as lighter bars and coverage as darker bars

Random < IPR < Disp < IPR+Disp
Results – Attack Behavior

Attack behavior varies on a single binary
Results – Attack Behavior

Attack behavior varies on a single binary

IPR attacks against Endgame

Binary 785728 | 30.0% Potency | 10 Trials
Results – Attack Behavior

Attack behavior varies on a single binary
Results – Attack Behavior

Attack behavior varies on a single binary
Results – Attack Behavior

Attack behavior varies on a single binary

Attack behavior varies between different binaries, depending on many variables
Results – Attack Behavior

Attack behavior varies on a single binary

Attack behavior varies between different binaries, depending on many variables
Results – Attack Behavior

Attack behavior varies on a single binary

Attack behavior varies between different binaries, depending on many variables
Results – Attack Behavior

Attack behavior varies on a single binary

Attack behavior varies between different binaries, depending on many variables
Results – Contrasting Attack Types

Attack success rates at each iteration in the white-box setting averaged over all target models and attacked binaries.
Results – Contrasting Attack Types

Attack success rates at each iteration in the white-box setting averaged over all target models and attacked binaries
Results – Contrasting Attack Types

Attack success rates at each iteration in the white-box setting averaged over all target models and attacked binaries
Attack success rates at each iteration in the white-box setting averaged over all target models and attacked binaries
Results – Contrasting Attack Types

Attack success rates at each iteration in the white-box setting averaged over all target models and attacked binaries.
Results – Effects on Anti-Viruses

Unmodified malicious binaries were detected by a median of 55/68 AVs
Results – Effects on Anti-Viruses

Unmodified malicious binaries were detected by a median of 55/68 AVs

Randomly transformed malicious binaries were detected by a median of 42/68 AVs

VirusTotal. https://www.virustotal.com/. Online
Results – Effects on Anti-Viruses

Unmodified malicious binaries were detected by a median of 55/68 AVs

Randomly transformed malicious binaries were detected by a median of 42/68 AVs

Adversarially transformed malicious binaries were detected by a median of 33-36/68 AVs

VirusTotal. https://www.virustotal.com/. Online
Potential Defenses

- Binary normalization – effective against IPR, ineffective against Displacement
Potential Defenses

- Binary normalization – effective against IPR, ineffective against Displacement
- Masking random instructions – effective when masking over 25% of instructions
Potential Defenses

• Binary normalization – effective against IPR, ineffective against Displacement
• Masking random instructions – effective when masking over 25% of instructions
• Adversarial training – currently not computationally feasible
Summary

• Described a process for modifying executable bytes of a binary to produce adversarial examples
  • Best attack succeeded in evading detection from all malware classification DNNs on nearly every binary

• Functionally preserving transformation code available on Github
  • Does not contain attack algorithm
  • https://github.com/pwwl/enhanced-binary-diversification

• Thank you for your time!
Malware Makeover: Breaking ML-based Static Analysis by Modifying Executable Bytes

Keane Lucas, Mahmood Sharif, Lujo Bauer, Michael K. Reiter, Saurabh Shintre