o-glassesX: Compiler Provenance Recovery with Attention Mechanism from a Short Code Fragment

Yuhei Otsubo*,†, Akira Otsuka†, Mamoru Mimura‡ †, Takeshi Sakaki§, and Hiroshi Ukegawa*

*National Police Agency, Tokyo, Japan
†Institute of Information Security, Kanagawa, Japan
‡National Defense Academy, Kanagawa, Japan
§The University of Tokyo, Tokyo, Japan
Introduction
Forensic Scientists
Computer Forensics

- Deleted files
- Malicious Documents
- Memory
- Email

Find Digital Evidences

Attackers

Victim PC

C2 Server
Author Identification

Compiler Provenance

- Compiler
- Family
- Version
- Optimization level

Static Link Libraries

Compile time

Malicious EXE

API name

Specific Strings

Secret key

Resources

Language
Multiple Compiler Binary

Make

What is the truth label of A.exe?
Multiple Compiler Binary

What is the truth label of A.exe?
- A Compiler?
- B Compiler?
Multiple Compiler Binary

What is the truth label of A.exe?
- A Compiler?
- B Compiler?
- X Compiler?
Multiple Compiler Binary

What is the truth label of `A.exe`?
- A Compiler?
- B Compiler?
- X Compiler?
- C Linker?
What is the truth label of $A.exe$?
- A Compiler?
- B Compiler?
- X Compiler?
- C Linker?

What is the truth label of $a.obj$?
- A Compiler

What is the truth label of $b.obj$?
- B Compiler

What is the truth label of $x.lib$?
- Hmm... I think VC, because MS provide it!
Multiple Compiler Binary

Easy to make the ground truth

What is the truth label of A.exe?
- A Compiler?
- B Compiler?
- X Compiler?
- C Linker?

What is the truth label of a.obj?
- A Compiler

What is the truth label of b.obj?
- B Compiler

What is the truth label of x.lib?
- Hmm... I think VC, because MS provide it!
Collect as much of the attacker's trace as possible even from fragmented files.
Preliminaries
**o-glasses**

x86 code (e.g., shellcode) detector [arXive.1806.05328]

- **Input**: 16 x86 instructions
- **Output**: Program or not
- **F1**: 0.9995

Executable file (compressed)

Grayscale

o-glasses (1d-CNN)

Correct data

Shell Code (splitted)
o-glasses

x86 code (e.g., shellcode) detector [arXive.1806.05328]

- Applying to compiler identification
- Black Box Problem
**Attention Is All You Need**

Łukasz Kaiser et al., NIPS, 2017

---

**Figure 1: The Transformer - model architecture.**

---

**Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.**

<table>
<thead>
<tr>
<th>Model</th>
<th>BLEU EN-DE</th>
<th>BLEU EN-FR</th>
<th>Training Cost (FLOPs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ByteNet [15]</td>
<td>23.75</td>
<td>39.2</td>
<td>1.0 \times 10^{20}</td>
</tr>
<tr>
<td>Deep-Att + PosUnk [32]</td>
<td>24.6</td>
<td>39.92</td>
<td>2.3 \times 10^{19}</td>
</tr>
<tr>
<td>GNMT + RL [31]</td>
<td>25.16</td>
<td>40.46</td>
<td>9.6 \times 10^{18}</td>
</tr>
<tr>
<td>ConvS2S [8]</td>
<td>26.03</td>
<td>40.56</td>
<td>2.0 \times 10^{19}</td>
</tr>
<tr>
<td>MoE [26]</td>
<td></td>
<td></td>
<td>1.2 \times 10^{20}</td>
</tr>
<tr>
<td>Deep-Att + PosUnk Ensemble [32]</td>
<td>26.30</td>
<td>40.4</td>
<td>1.8 \times 10^{20}</td>
</tr>
<tr>
<td>GNMT + RL Ensemble [31]</td>
<td>26.36</td>
<td>41.16</td>
<td>1.1 \times 10^{21}</td>
</tr>
<tr>
<td>ConvS2S Ensemble [8]</td>
<td>26.36</td>
<td><strong>41.29</strong></td>
<td>7.7 \times 10^{19}</td>
</tr>
<tr>
<td>Transformer (base model)</td>
<td>27.3</td>
<td>38.1</td>
<td><strong>3.3 \times 10^{18}</strong></td>
</tr>
<tr>
<td>Transformer (big)</td>
<td><strong>28.4</strong></td>
<td><strong>41.0</strong></td>
<td><strong>2.3 \times 10^{19}</strong></td>
</tr>
</tbody>
</table>
Basic of Attention

Input [q_length, depth]

Memroy [m_length, depth]

Query [q_length, depth]

Key [m_length, depth]

matmul

Softmax

Output [q_length, depth]

Value [m_length, depth]

matmul

Att. W [q_length, m_length]
Basic of Attention

query = 'key2'
memory = {'key1': 'value2',
          'key2': 'value2',
          'key3': 'value3',
          'key4': 'value4'}
memory[query] = 'value2'
query = 'key2'
memory = {'key1':'value2',
          'key2':'value2',
          'key3':'value3',
          'key4':'value4'}
memory[query] = 'value2'
Basic of Attention

query = 'key2'
memory = {'key1':'value2',
          'key2':'value2',
          'key3':'value3',
          'key4':'value4'}
memory[query] = 'value2'
Basic of Attention

query = 'key2'
memory = {'key1':'value2',
         'key2':'value2',
         'key3':'value3',
         'key4':'value4'}
memory[query] = 'value2'
Dot-Product Attention vs. Dictionary Object

query = 'key2'
memory = {'key1': 'value2',
          'key2': 'value2',
          'key3': 'value3',
          'key4': 'value4'}
memory[query] = 'value2'
Self-Attention

Input $[q_{\text{length}}, \text{depth}]$

Query $[q_{\text{length}}, \text{depth}]$

Key $[m_{\text{length}}, \text{depth}]$

matmul

Softmax

Att. W $[q_{\text{length}}, m_{\text{length}}]$

Input $[m_{\text{length}}, \text{depth}]$

Value $[m_{\text{length}}, \text{depth}]$

matmul

Output $[q_{\text{length}}, \text{depth}]$
Positional Encoding (PE)

PE (Positional Encoding) adds information about the word position to the input word vectors for learning the context of words.

\[ Y(X) = X + \alpha PE \]

\[ PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right) \]

\[ PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right) \]
Proposed Method
o-glassesX

Convert

Binary

x86 instructions

128-bit length instructions

CNN

PE

Att. Input

Query

matmul

Softmax

Att. W

matmul

Att. Output

BN

FFN

Softmax

CNN

CNN

Key

Value

Same as o-glasses
Preprocessing details

Binary

```
60 B9 67 01 00 00 EB 0F
```

x86 instructions

```
PUSHA
MOV ECX, 0x167
JMP loc_17
```

128-bit length instructions

```
60 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B9 67 01 00 00 00 00 00 00 00 00 00 00 00 00 00
EB 0F 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

16 bytes

```
00 00 11 00 00 00 00 00 00 00 00 00 00 00 00 00
```

128 bits

```
00 00 00 11 00 00 00 00 00 00 00 00 00 00 00 00
10 01 11 01 11 11 11 00 01 10 00 00 00 00 00 00
11 01 10 11 11 11 11 11 11 00 00 00 00 00 00 00
```

Convert

```
Binary
```

```
x86 instructions
```

```
128-bit length instructions
```

CNN

Same as o-glasses
The 1st CNN Layer

Each unit in CNN has specially local connections to the input units, called a Kernel. Every kernel shares the weight parameters with the others in the same layer.

Each kernel covers a single instruction by adjusting the hyperparameters.

Kernel size = 128

Instruction vector

Depth (=96)

Instruction vector

Instruction vector

Instruction vector

Instruction vector

128-bit length instruction

128-bit length instruction

128-bit length instruction

128-bit length instruction

128-bit length instruction

128-bit length instruction

CNN

Convert

Binary

x86 instructions

128-bit length instructions

Same as o-glasses
Evaluation
### Training Dataset

Collecting source code files from GitHub
Compiling various compilers and options

<table>
<thead>
<tr>
<th>Total</th>
<th>19 labels</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compiler</td>
<td>4 families</td>
</tr>
<tr>
<td>Visual C++, GCC, Clang and Intel C++ Compiler</td>
<td></td>
</tr>
<tr>
<td>Opt. level</td>
<td>2 types</td>
</tr>
<tr>
<td>maximum or not</td>
<td></td>
</tr>
<tr>
<td>CPU Arc.</td>
<td>2 types</td>
</tr>
<tr>
<td>x86 or x86-64</td>
<td></td>
</tr>
</tbody>
</table>

**TABLE II.**  D A T A S E T  C O M P I L A T I O N  S E T T I N G S.

<table>
<thead>
<tr>
<th>Compiler family</th>
<th>Version</th>
<th>Architecture</th>
<th>Optimization Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>VC</td>
<td>2003</td>
<td>x86</td>
<td>-Od, -Ox</td>
</tr>
<tr>
<td>GCC</td>
<td>2017</td>
<td>x86, x86-64</td>
<td>-Od, -Ox</td>
</tr>
<tr>
<td>Clang</td>
<td>6.3.0</td>
<td>x86, x86-64</td>
<td>-O0, -O3</td>
</tr>
<tr>
<td>Clang</td>
<td>5.0.2</td>
<td>x86, x86-64</td>
<td>-O0, -O3</td>
</tr>
<tr>
<td>ICC</td>
<td>19.0.0.117</td>
<td>x86-64</td>
<td>-O0, -O3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Label</th>
<th>#Binaries</th>
<th>#Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>VC17,32,none(Od)</td>
<td>1,170</td>
<td>369,605</td>
</tr>
<tr>
<td>VC17,32,max(Ox)</td>
<td>1,147</td>
<td>255,143</td>
</tr>
<tr>
<td>VC17,64,none(Od)</td>
<td>1,456</td>
<td>540,568</td>
</tr>
<tr>
<td>VC17,64,max(Ox)</td>
<td>1,242</td>
<td>542,020</td>
</tr>
<tr>
<td>VC03,32,none(Od)</td>
<td>1,350</td>
<td>292,277</td>
</tr>
<tr>
<td>VC03,32,max(Ox)</td>
<td>1,306</td>
<td>270,743</td>
</tr>
<tr>
<td>GCC,32,none(O0)</td>
<td>2,111</td>
<td>227,004</td>
</tr>
<tr>
<td>GCC,32,max(O3)</td>
<td>1,844</td>
<td>239,821</td>
</tr>
<tr>
<td>GCC,64,none(O0)</td>
<td>1,582</td>
<td>283,276</td>
</tr>
<tr>
<td>GCC,64,max(O3)</td>
<td>1,580</td>
<td>287,775</td>
</tr>
<tr>
<td>Clang,32,none(O0)</td>
<td>1,205</td>
<td>101,024</td>
</tr>
<tr>
<td>Clang,32,max(O3)</td>
<td>1,196</td>
<td>86,521</td>
</tr>
<tr>
<td>Clang,64,none(O0)</td>
<td>1,892</td>
<td>332,278</td>
</tr>
<tr>
<td>Clang,64,max(O3)</td>
<td>1,883</td>
<td>246,500</td>
</tr>
<tr>
<td>ICC,32,none(Od)</td>
<td>1,761</td>
<td>1,494,677</td>
</tr>
<tr>
<td>ICC,32,max(Ox)</td>
<td>1,724</td>
<td>1,161,499</td>
</tr>
<tr>
<td>ICC,64,none(Od)</td>
<td>1,796</td>
<td>1,419,705</td>
</tr>
<tr>
<td>ICC,64,max(Ox)</td>
<td>1,728</td>
<td>1,046,958</td>
</tr>
<tr>
<td>Others</td>
<td>101</td>
<td>912,855</td>
</tr>
<tr>
<td>Total</td>
<td>28,074</td>
<td>10,110,249</td>
</tr>
<tr>
<td>Train</td>
<td>Predict</td>
<td>VC</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------</td>
<td>----------</td>
</tr>
<tr>
<td>VC03,32,none(Od)</td>
<td>1  0.9604 0.0355 0.0026 0.0002 0.0002 0.0002 0.0001 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000</td>
<td></td>
</tr>
<tr>
<td>VC17,32,none(Od)</td>
<td>2  0.0463 0.9517 0.0002 0.0007 0.0001 0.0001 0.0001 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000</td>
<td></td>
</tr>
<tr>
<td>VC03,32,max(Ox)</td>
<td>3  0.0026 0.0011 0.9875 0.0061 0.0001 0.0003 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000</td>
<td></td>
</tr>
<tr>
<td>VC17,32,max(Ox)</td>
<td>4  0.0004 0.0010 0.1444 0.9774 0.0001 0.0005 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000</td>
<td></td>
</tr>
<tr>
<td>VC17,64,none(Od)</td>
<td>5  0.0002 0.0001 0.0002 0.0001 0.9978 0.0008 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000</td>
<td></td>
</tr>
<tr>
<td>VC17,64,max(Ox)</td>
<td>6  0.0002 0.0001 0.0004 0.0004 0.0013 0.9931 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000</td>
<td></td>
</tr>
<tr>
<td>GCC,32,none(O0)</td>
<td>7  0.0002 0.0000 0.0001 0.0000 0.0000 0.0000 0.9973 0.0009 0.0004 0.0001 0.0000 0.0000 0.0000 0.0000</td>
<td></td>
</tr>
<tr>
<td>GCC,32,max(O3)</td>
<td>8  0.0001 0.0000 0.0005 0.0002 0.0000 0.0000 0.011 0.9921 0.0000 0.0003 0.0000 0.0000 0.0000 0.0000</td>
<td></td>
</tr>
<tr>
<td>GCC,64,none(O0)</td>
<td>9  0.0001 0.0001 0.0000 0.0000 0.0000 0.0000 0.008 0.9970 0.0000 0.0006 0.0000 0.0000 0.0000 0.0000</td>
<td></td>
</tr>
<tr>
<td>GCC,64,max(O3)</td>
<td>10 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.001 0.9976 0.0000 0.0002 0.0000 0.0000 0.0000 0.0000</td>
<td></td>
</tr>
<tr>
<td>Clang,32,none(O0)</td>
<td>11 0.0003 0.0002 0.0000 0.0000 0.0000 0.0000 0.004 0.0004 0.0004 0.0000 0.0001 0.0000 0.0000 0.0000</td>
<td></td>
</tr>
<tr>
<td>Clang,32,max(O3)</td>
<td>12 0.0001 0.0000 0.0006 0.0011 0.0000 0.0001 0.004 0.9986 0.0000 0.0003 0.0000 0.0000 0.0000 0.0000</td>
<td></td>
</tr>
<tr>
<td>Clang,64,none(O0)</td>
<td>13 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.000 0.9973 0.0000 0.0003 0.0000 0.0000 0.0000 0.0000</td>
<td></td>
</tr>
<tr>
<td>Clang,64,max(O3)</td>
<td>14 0.0000 0.0000 0.0001 0.0001 0.0001 0.0006 0.000 0.9976 0.0000 0.0002 0.0000 0.0000 0.0000 0.0000</td>
<td></td>
</tr>
<tr>
<td>ICC,32,none(Od)</td>
<td>15 0.0001 0.0000 0.0001 0.0000 0.0000 0.0000 0.002 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000</td>
<td></td>
</tr>
<tr>
<td>ICC,32,max(Ox)</td>
<td>16 0.0002 0.0001 0.0029 0.0028 0.0000 0.0001 0.000 0.0003 0.0000 0.0001 0.0000 0.0000 0.0000 0.0000</td>
<td></td>
</tr>
<tr>
<td>ICC,64,none(Od)</td>
<td>17 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000</td>
<td></td>
</tr>
<tr>
<td>ICC,64,max(Ox)</td>
<td>18 0.0000 0.0000 0.0001 0.0001 0.0001 0.0016 0.000 0.0001 0.0000 0.0001 0.0000 0.0000 0.0000 0.0000</td>
<td></td>
</tr>
<tr>
<td>Others</td>
<td>19 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000</td>
<td></td>
</tr>
</tbody>
</table>
# Comparison of Performance of Related Work

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Accuracy</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>All components</td>
<td>.9884 (19)</td>
<td>.9555 (19)</td>
<td>.9421 (19)</td>
<td>.9323 (19)</td>
<td>.924 (3)</td>
<td>.604 (18)</td>
<td>.918 (18)</td>
<td>.801 (6)</td>
<td></td>
</tr>
<tr>
<td>Compiler family</td>
<td>.9886 (4)</td>
<td>.9670 (4)</td>
<td>.9140 (4)</td>
<td>.9271 (4)</td>
<td>.924 (3)</td>
<td>.983 (3)</td>
<td>.999 (3)</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Optimization</td>
<td>.9989 (2)</td>
<td>.9943 (2)</td>
<td>.9830 (2)</td>
<td>.9864 (2)</td>
<td>-</td>
<td>.971 (2)</td>
<td>.999 (2)</td>
<td>.917 (2)</td>
<td></td>
</tr>
<tr>
<td>Architecture</td>
<td>.9997 (2)</td>
<td>.9985 (2)</td>
<td>.9959 (2)</td>
<td>.9940 (2)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Code/Non-code</td>
<td>.9999 (2)</td>
<td>.9995 (2)</td>
<td>.9991 (2)</td>
<td>.9987 (2)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td><strong>ML model</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Features</td>
<td>64 Instructions</td>
<td>16 Instructions</td>
<td>64 Instructions</td>
<td>16 Instructions</td>
<td>Byte Seq.</td>
<td>1 Function</td>
<td>Function Seq.</td>
<td>1 File</td>
<td>(k-means)</td>
</tr>
<tr>
<td>#Samples</td>
<td>1,793,478</td>
<td>1,900,000</td>
<td>471,124</td>
<td>1,886,521</td>
<td>81,886,169</td>
<td>955,000</td>
<td>955,000</td>
<td>1,177</td>
<td></td>
</tr>
<tr>
<td>#Binaries</td>
<td>28,074</td>
<td>28,074</td>
<td>28,074</td>
<td>28,074</td>
<td>1,119</td>
<td>2,686</td>
<td>2,686</td>
<td>1,177</td>
<td></td>
</tr>
<tr>
<td><strong>Dataset</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VC</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>1</td>
<td>6</td>
<td>6</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>GCC</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>8</td>
<td>8</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Clang</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>ICC</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Non-code</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td><strong>K-fold cross-validation</strong></td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>(Anomalous)</td>
<td>(Anomalous)</td>
<td>(Anomalous)</td>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>
Calculating ‘Why’ with the Attention Mechanism

Algorithm 1 Automatic feature extraction procedure

1: InputFile ← Compile(SourceFile, Optimization).pullCode()
2: Offset ← 0
3: while Offset < InputFile.EndOffset do
4: Input ← InputFile.pull(Offset, L)
5: Result ← model.predictor(Input)
6: if Result.Label = RealLabel then
7: if Result.Confidence > 0.99 then
8: i ← argmax(L2(Result.AttentionWeight))
9: Count[input[i]] ++
10: end if
11: end if
12: Offset ++
13: end while
14: Print(Ranking(Count))
Typical Instructions for each compiler

e.g., when focusing on the NOP instruction...

Fig. 5. Typical instructions for each compiler against aes.c
# Case Study: Various Optimization Levels

## Original Code (no optimization)

<table>
<thead>
<tr>
<th>Function name</th>
<th>Segment</th>
<th>Start</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>sub_401010</td>
<td>.text</td>
<td>00401010</td>
<td>0000014C</td>
</tr>
<tr>
<td>sub_401160</td>
<td>.text</td>
<td>00401160</td>
<td>0000023A</td>
</tr>
<tr>
<td>sub_4013A0</td>
<td>.text</td>
<td>004013A0</td>
<td>00000147</td>
</tr>
<tr>
<td>sub_4014F0</td>
<td>.text</td>
<td>004014F0</td>
<td>000000BF</td>
</tr>
<tr>
<td>sub_4015B0</td>
<td>.text</td>
<td>004015B0</td>
<td>0000039D</td>
</tr>
<tr>
<td>_main</td>
<td>.text</td>
<td>00401950</td>
<td>00004129</td>
</tr>
<tr>
<td>sub_405AA0</td>
<td>.text</td>
<td>00405AA0</td>
<td>00000046</td>
</tr>
<tr>
<td>sub_405AF0</td>
<td>.text</td>
<td>00405AF0</td>
<td>00000085</td>
</tr>
<tr>
<td>sub_405B80</td>
<td>.text</td>
<td>00405B80</td>
<td>0000042D</td>
</tr>
<tr>
<td>sub_405FB0</td>
<td>.text</td>
<td>00405FB0</td>
<td>000001E7</td>
</tr>
<tr>
<td>sub_4061A0</td>
<td>.text</td>
<td>004061A0</td>
<td>00000015</td>
</tr>
<tr>
<td>sub_4061CD</td>
<td>.text</td>
<td>004061CD</td>
<td>0000002D</td>
</tr>
<tr>
<td>MakeMnemonic</td>
<td>.text</td>
<td>004061EE</td>
<td>00000006</td>
</tr>
<tr>
<td>InstrDasm</td>
<td>.text</td>
<td>004061F4</td>
<td>00000006</td>
</tr>
<tr>
<td>InstrDecode</td>
<td>.text</td>
<td>004061FA</td>
<td>00000006</td>
</tr>
<tr>
<td>_crtCorExitProcess</td>
<td>.text</td>
<td>00406200</td>
<td>0000002B</td>
</tr>
<tr>
<td>__crtExitProcess</td>
<td>.text</td>
<td>0040622B</td>
<td>00000017</td>
</tr>
<tr>
<td>__lockexit</td>
<td>.text</td>
<td>00406243</td>
<td>00000009</td>
</tr>
<tr>
<td>__unlockexit</td>
<td>.text</td>
<td>0040624C</td>
<td>00000009</td>
</tr>
<tr>
<td>__init_pointers</td>
<td>.text</td>
<td>00406255</td>
<td>00000033</td>
</tr>
<tr>
<td>__interm_e</td>
<td>.text</td>
<td>00406288</td>
<td>00000024</td>
</tr>
<tr>
<td>__cinit</td>
<td>.text</td>
<td>004062AC</td>
<td>00000097</td>
</tr>
<tr>
<td>_doexit</td>
<td>.text</td>
<td>00406343</td>
<td>00000140</td>
</tr>
<tr>
<td>_exit</td>
<td>.text</td>
<td>00406483</td>
<td>00000016</td>
</tr>
<tr>
<td>__exit</td>
<td>.text</td>
<td>00406499</td>
<td>00000016</td>
</tr>
<tr>
<td>__cexit</td>
<td>.text</td>
<td>004064AF</td>
<td>0000000F</td>
</tr>
<tr>
<td>__c_exit</td>
<td>.text</td>
<td>004064BE</td>
<td>0000000F</td>
</tr>
<tr>
<td>__amsg_exit</td>
<td>.text</td>
<td>004064CD</td>
<td>0000001D</td>
</tr>
<tr>
<td>_printf</td>
<td>.text</td>
<td>004064EB</td>
<td>000000A7</td>
</tr>
</tbody>
</table>

## Static Link Library (maximum optimization)

---

Office MalScanner.exe (OMS)
Bit-Image of OMS

- NULL (0x00)
- Control Characters (0x01-0x1F)
- Printable Characters (0x20-0x7F)
- Others (0x80-0xFF)

Stirling[^10]: a hex editor

Visualization of OMS by o-glassesX

- Visual C++
- GCC
- Clang
- Intel C++ Compiler

Program Code:
- VC03,32,none
- VC17,32,none
- VC03,32,max
- VC17,32,max
- VC17,64,none
- VC17,64,max
- GCC,32,none
- GCC,32,max
- GCC,64,none
- GCC,64,max
- Clang,32,none
- Clang,32,max
- Clang,64,none
- Clang,64,max
- ICC,32,none
- ICC,32,max
- ICC,64,none
- ICC,64,max
- Others
Visualization of OMS by o-glassesX

Program Code:
- VC03,32,max
- VC17,64,max
- GCC,32,max
- Clang,32,max
- ICC,32,max

Optimization Levels:
- Low Optimization Level
- High Optimization Level
Case Study: Tracking Emdivi RATs in Dev. Env.

EMDIVI malware family is used in targeted email attacks against Japanese organizations. It allows machines to be remotely controlled by attackers for malicious commands and other activities.

Japan pension system hacked, 1.25 million cases of personal data leaked

Blue Termite cyber-espionage campaign targets hundreds of organizations in Japan

The attackers spread through exploiting a zero-day Flash player exploit and a sophisticated backdoored exploit, customized to each victim. They have been active at least since 2013.
The Version of Emdivi

- Almost attached malware is compiled just before used
  So, the sender and the developer may be in the same group

- Frequently Updated
  t17  : For initial compromise
  t19,t20 : For expanding the intrusion (High stealth performance)
Emdivi dataset
Analysis Report made by Macnica Networks
https://www.macnica.net/security/report_01.html/

163 MD5 Hashes of the Emdivi Family
Difference of Emdivi Rats in Dev. Env.

All Sample
Architecture : 32-bit (x86)
Compiler family : Visual C++
Optimization level : max

Focusing on Compiler Version
Yellow : relatively new compiler
Blue : relatively old compiler

Type A: Yellow
Type B: Blue -> Yellow
Type C: Yellow -> Blue -> Yellow

VC03,32,none
VC17,32,none
**VC03,32,max**
**VC17,32,max**
VC17,64,none
VC17,64,max
Compile-time and version of Emdivi RATs

Type A

Type B

Type C

For initial compromise

For expanding the intrusion

5 months

3 months
Limitation

Obfuscated Code
Input machine code need to be already de-obfuscated.

Multi CPU Architectures
This method may be applied to many CPU architecture besides x86. Splitting binary by instruction is difficult in two more CPU architectures inputs at the same time. This limitation will be resolved, if new bin2vec method supporting multi CPU Arc. is released…
Conclusion

High Recognition Rate for Stripped Machine Code
16-instruction input: .956 accuracy
64-instruction input: .988 accuracy

Solution to Black Box Problem
o-glassesX can calculate how much input data contributes to output in units of instructions

Case Study: Emdivi
It has been revealed that there are three attackers in the same attack group.

o-glassesX and our dataset are available at
https://github.com/yotsubo/o-glassesX