CUBISMO: Decloaking Server-side Malware via Cubist Program Analysis

Abbas Naderi-Afooshteh, Yonghwi Kwon, Anh Nguyen-Tuong, Mandana Bagheri-Marzijarani, and Jack W. Davidson

University of Virginia
Cubist art analyzes multiple aspects of an object, breaks them down, and reassembles them for presentation.

Pablo Picasso, 1910
Girl with a Mandolin (Fanny Tellier)
Oil on canvas, 100.3 x 73.6 cm
Museum of Modern Art, New York
Server-side (PHP) malware

(a) Normalized Program

```php
<?php
error_reporting(0);
@ini_set('error_log',NULL);
@ini_set('log_errors',0);
@ini_set('display_errors','Off');
@
eval(base64_decode('aWYobWQ1KCRfUE9TVFsicGYiXSkgPT09I...DQ4YzJOeWFYQi4uLkwySnZaSGsrUEM5b2RHMXNQZzBLIikpOw=='));
@
ini_restore('error_log');
@
ini_restore('display_errors');
...
```

(b) Deobfuscated Program 1

```php
<?php
error_reporting(0);
@ini_set('error_log',NULL);
@ini_set('log_errors',0);
@ini_set('display_errors','Off');
if(md5($_POST['pf']) === '...')
    eval(base64_decode($_POST['...']));
...
if($patchedfv === '...') {
    @ob_end_clean();
    die;
}
$uERCz = '+= ZXS...>68,Q;';
$kcxtOJojkr = $uERCz('8ZfC<...==72-XE08...RA715e<Ei>Z5M83fSbQ:0');
$kcxtOJojkr();
@ini_restore('error_log');
@ini_restore('display_errors');
...
```

(c) Deobfuscated Program 2

```php
<?php
error_reporting(0);
@ini_set('error_log',NULL);
@ini_set('log_errors',0);
@ini_set('display_errors','Off');
if(md5($_POST['pf']) === '...')
    eval(base64_decode($_POST['...']));
...
if($patchedfv === '...') {
    @ob_end_clean();
    die;
}
```

CUBISMO, 2019

PHP Malware and Its Multiple Aspects, Deobfuscation in PHP, ACSAC'19
Multiple aspects of web server malware (i.e., PHP malware)

- PHP is a dynamic language, making web development easy, so as malware development

  1. Evasive Code
     - Decide whether to run or not, depending on the context

  2. Multiple Layers of Obfuscation via Dynamic Constructs
     - Use `eval` and `include` to dynamically generate/include code
       - Obfuscation is cheap and easy in PHP

  3. Automated Variant Generation
     - Creating variants of PHP malware is easy
Evasive and Multiple Layers of Obfuscation

if( .. = $pass)
    die("...");

Loop head

if( $secret ... )
    eval($obfuscated_code)

include($file)

eval($obfuscated_code) defines $filename, and include($filename) will update $obfuscated_code

1 if ($_GET[1]!=$password)
2     die("Nothing to see here.");
3 for (...)
4     if ($secret === "...") {
5         include($filename);
6     } else {
7         eval($obfuscated_code);
8     }
if ($SECRET[1]!=$password)
    die("Nothing to see here.");
for (...)
    if ($SECRET === "...") {
        include($filename);
    } else {
        eval($obfuscated_code);
    }
2. Multiple Layers of Obfuscation

```php
if ($_GET[1] != $password)
    die("Nothing to see here.");
for (...) {
    if ($secret === "...") {
        include($filename);
    } else {
        eval($obfuscated_code);
    }
} else {
    eval($obfuscated_code);
}
```
2. Multiple Layers of Obfuscation

**Deobfuscation Layer 1**

```
if ($_GET[1]!=$password)
    die("Nothing to see here.");
for (...)
    if ($secret === "...") {
        include($filename);
    } else {
        eval($obfuscated_code);
    }
```
2. Multiple Layers of Obfuscation

```php
if ($_GET[1]!=$password)
    die("Nothing to see here.");
for (...)
    if ($secret === "...") {
        include($filename);
    } else {
        eval($obfuscated_code);
    }
```

Deobfuscation Layer 2
3. Automated Malware Variant Generation

• Creating PHP malware variants is as simple as a string manipulation

<table>
<thead>
<tr>
<th></th>
<th>Original Malware</th>
<th>Malware Variant 1</th>
<th>Malware Variant 2</th>
<th>Malware Variant 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$s_pass = '4b34f78fbd220513438011562320d47f';</td>
<td>$s_pass = 'b4616d42a983401bcf344f9c18675777';</td>
<td>$s_pass = '62908bf72c21a3d8eea23a55dec98e4b';</td>
<td>$s_pass = '62908bf72c21a3d8eea23a55dec98e4b';</td>
</tr>
<tr>
<td>2</td>
<td>$x=gzinflate(base64_decode('7b1pe+0415j80fM88x8U3XpT5chVXLv1LR1KFEv715QnWv/3EmJm7TuFvnL0BsmzLDl...+JDLu+vGCe/m0F3+e7PpQzuf97sMN0MIA7DsAeZPX/5/'));</td>
<td>eval('$x=gzi.'.'flate('base'.'.4_de'.').code('7b1pe+0415j80fM88x8U3XpT5chVXLv1LR1KFEv715QnWv/3EmJm7TuFvnL0BsmzLDl...+JDLu+vGCe/m0F3+e7PpQzuf97sMN0MIA7DsAeZPX/5/'));</td>
<td>eval('$x=g.'.'zin'.').flate('base'.').4_de'.').ode('7b1pe+0415j80fM88x8U3XpT5chVXLv1LR1KFEv715QnWv/3EmJm7TuFvnL0BsmzLDl...+JDLu+vGCe/m0F3+e7PpQzuf97sMN0MIA7DsAeZPX/5/'));</td>
<td>eval('$x=g'.).zip();</td>
</tr>
</tbody>
</table>
3. Automated Malware Variant Generation

- Changing $s\_pass$

<table>
<thead>
<tr>
<th></th>
<th>Original Malware</th>
<th>Malware Variant 1</th>
<th>Malware Variant 2</th>
<th>Malware Variant 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$s_pass = '4b34f78fbd220513438011562320d47f';$</td>
<td>$s_pass = 'b4616d42a983401bcf344f9c18675777';$</td>
<td>$s_pass = '62908bf72c21a3d8eaa23a55dec98e4b';$</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>eval('?&gt;'.$x);</td>
<td>eval('?&gt;'.$x);</td>
<td>eval('?&gt;'.$x);</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>eval('?&gt;'.$x);</td>
<td>eval('?&gt;'.$x);</td>
<td>eval('?&gt;'.$x);</td>
<td></td>
</tr>
</tbody>
</table>

(a) Original Malware

(b) Malware Variant 1

(c) Malware Variant 2

(d) Malware Variant 3
3. Automated Malware Variant Generation

- "$x = \text{gzinflate(base64_decode}$" $\Rightarrow$ "eval('$x=\text{gzip}.{fla..."

1. $\text{s_pass} = '4b34f78fd220513438011562320d47f';$
2. $x = \text{gzinflate(base64_decode}(7b1pe+04i5j80fM88x8U3XpT5chVXLV1RIKFEtv1L5QnWv/3EmJm7TuFnvLOBSzmLd1...+JDlu+vGCE/m0F3+e7PpQzuf97sMY80MIA7DsAeZPX/5/'));$
3. eval('?>'.$x);

(a) Original Malware

(b) Malware Variant 1

10 $\text{s_pass} = '4b34f78fd220513438011562320d47f';$
11 eval('x=\text{gzinflate(base64_decode(7b1p e+04i5j80fM88x8U3XpT5chVXLV1RIKFEtv1L5QnWv/3EmJm7TuFnvLOBSzmLd1...+JDlu+vGCE/m0F3+e7PpQzuf97sMY80MIA7DsAeZPX/5/'))');$
12 eval('?>'.$x);

(c) Malware Variant 2

20 $\text{s_pass} = 'b4616d42a983401bcf344f9c18675777';$
21 eval('$x=\text{gzinflate(base64_decode(7b1p e+04i5j80fM88x8U3XpT5chVXLV1RIKFEtv1L5QnWv/3EmJm7TuFnvLOBSzmLd1...+JDlu+vGCE/m0F3+e7PpQzuf97sMY80MIA7DsAeZPX/5/'))');$
22 eval('?>'.$x);

(d) Malware Variant 3
Overview: CUBISMO

(a) CUBISMO

Input (1 file) PHP Program

CUBISMO

Output (N files) Multiple Decoded PHP Programs

(b) Leveraging Existing Tools

Existing Malware Detectors

Detect Malware if at least 1 out of N file is flagged

Normalization

Normalized Abstract Syntax Tree

CUBISMO

Counter-factual Execution

Code Generator

Decloacked ASTs
Exposing Multiple Aspects of Malware

```php
if ($_GET[1]!=$password)
    die("Nothing to see here.");
for (...)
    if ($secret === "...") {
        include($filename);
    } else {
        eval($obfuscated_code);
    }
```
Counter-factual Execution [MalMax, CCS’19]

```php
if ($_GET[1]!=$password)
    die("Nothing to see here.");
for (...)
    if ($secret === "...") {
        include($filename);
    } else {
        eval($obfuscated_code);
    }
```
if( .. = $pass)

die("...");

if( $secret ...

include($file)

eval($obfusc)

Loop head

1 if ($_GET[1]!=$password)
2 die("Nothing to see here.");
3 for (...)
4 if ($secret === "...") {
5 include($filename);
6 } else {
7 eval($obfuscated_code);
8 }
More details in the paper

• **Counter-factual Execution.** Exploring hidden malicious paths and execution contexts.

• **Sharing Global Artifacts between Paths.** Facilitating discovery of new dynamic code generation dependent on global artifacts (e.g., global variables).

• **Sandboxing.** Preventing malicious programs from harming the host system.

and more...
Evaluation: Dataset Collection

- **Real-world Website Deployments:** 400K real-world website snapshots deployed in the wild (via CodeGuard).
- **Nightly Backup:** Every night, a website is backed up when maldet finds one or more malware. Multiple versions of a website can be backed up.

Backup the entire website

if malware found
Evaluation: Numbers

• From **400K** website snapshots (about 3M files)
• **700K** files containing PHP code
• **1,269** files with dynamic constructs (potentially obfuscated)
  • **1,040** unique files.
• We scan them with VirusTotal: **688** files were detected.
• We manually analyze the remaining **352** files left undetected (with our previous work in CCS’19)
• Identified **56** previously undetected malware
• CUBISMO can reveal **53** out of the **56** malware samples
Evaluation: Methodology

• We use **VirusTotal** (as an existing tool in our pipeline)
  • We feed malware to CUBISMO that produces multiple decloaked files
  • (a) We feed the decloaked files to VT
  • (b) We also feed the original file to VT and then we compare (a) and (b)

• **VirusTotal learns!** and we consider that
  • After a few days of our submissions, VT starts to detect what they did not detect
  • Our experiments are less likely affected by this, because for each submission, we submit all the files generated from an original sample within a minute.
Evaluation: Why Though?

• Do Multiple Layers of Obfuscation Matter?
• Why not simply deobfuscate everything and then scan?

Naive Obfuscation

Advanced Obfuscation
Evaluation: Every Layer Matters

Original File
One decoder is observed

Code (Not obfuscated)

Decoder 1

Obfuscated Code 1

Deobfuscated File (First Layer)
Two decoders are observed

Code (Not obfuscated)

Decoder 1

Deobfuscated Code 1

Decoder 2

Obfuscated Code 2

Deobfuscated File (Second Layer)
One decoder is observed

Code (Not obfuscated)

Decoder 1

Deobfuscated Code 1

Deobfuscated Code 2

Removal of Decoder 2

Signature of Decoder 2

Deobfuscation

Anti-virus tools do not recognize this malware

Anti-virus tools flag this as malware by recognizing Decoder 2
# Evaluation: Everything Matters

<table>
<thead>
<tr>
<th>Orig.</th>
<th>Norm.</th>
<th>Layer 1</th>
<th>Layer 2</th>
<th>Layer 3</th>
<th>Layer 4</th>
<th>Orig.</th>
<th>Norm.</th>
<th>Layer 1</th>
<th>Layer 2</th>
<th>Layer 3</th>
<th>Layer 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>m1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>-</td>
<td>-</td>
<td>m29</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>m2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>m30</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>m3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>m31</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>m4</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>m32</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>m5</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>-</td>
<td>m33</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>m6</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>m34</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>m7</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>-</td>
<td>m35</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>m8</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>m36</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>m9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-</td>
<td>m37</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>m10</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>m38</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>m11</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>-</td>
<td>m39</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>m12</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>m40</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>m13</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>m41</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>m14</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>-</td>
<td>m42</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>m15</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>m43</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>m16</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>-</td>
<td>m44</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>m17</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>-</td>
<td>m45</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>m18</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>m46</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>m19</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>m47</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>m20</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>m48</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>m21</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>m49</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>m22</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>-</td>
<td>m50</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>m23</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>m51</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>m24</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>m52</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>m25</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>-</td>
<td>-</td>
<td>m53</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>m26</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>-</td>
<td>m54</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>m27</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-</td>
<td>m55</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>m28</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>m56</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

23
Evaluation: Details

- **False positive**: We test 100 benign PHP files with obfuscations (they do that to protect their code) and 200 benign PHP files from benign PHP applications.

- **Performance**: Decloaking process will be adding ~130% runtime overhead. We can parallelize the technique to improve the performance. Details in paper.

- And more in the paper.
Limitations

- Normalization would miss malicious code hidden in comments.

(a) Original Program
(PHP-Parser Crash)

(b) Normalized Program
(No Crash)
Thanks!

• CUBISMO is publicly available: https://cubismo.s3.amazonaws.com/cubismo.html

• Sponsors:
  • AFRL (FA8750-17-S-7007)
  • NSF (1916499 and 1850392)
  • CodeGuard