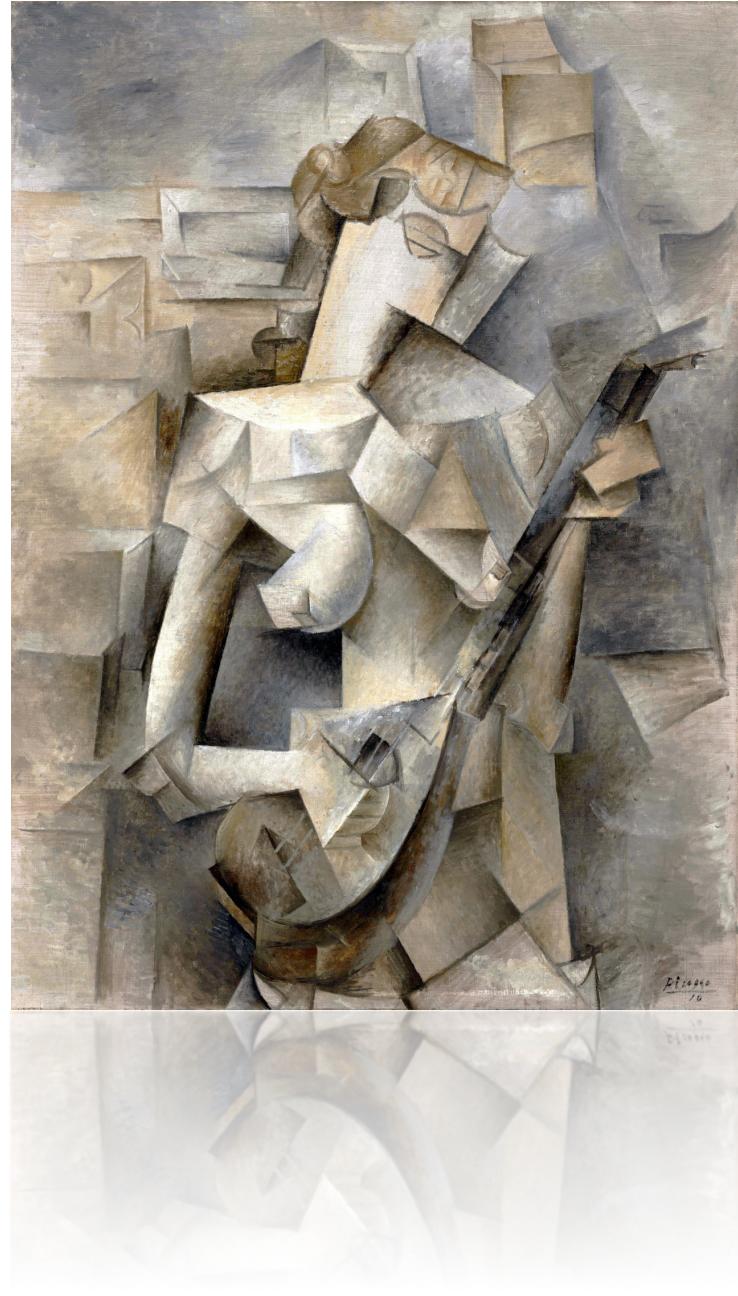


# CUBISMO: Decloaking Server-side Malware via Cubist Program Analysis

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

**University of Virginia**



# Cubist Art

“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

```
1  <?php  
2  error_reporting(0);  
3  @ini_set('error_log',NULL);  
4  @ini_set('log_errors',0);  
5  @ini_set('display_errors','Off');  
6  @eval(base64_decode('awYobWQ1KCRfUE9TVFsicGYiXSkgPT09I  
CIuLi...DQ4YzJ0eWFYQi4uLkwySnZaSG  
srUEM5b2RHMNXNQZzBLIikp0w=='));  
7  @ini_restore('error_log');  
8  @ini_restore('display_errors');  
9  ...
```

```
10 <?php  
11 error_reporting(0);  
12 @ini_set('error_log',NULL);  
13 @ini_set('log_errors',0);  
14 @ini_set('display_errors','Off');  
15 if(md5($_POST["pf"]) === "...")  
16     eval(base64_decode($_POST["..."]));  
17 ...  
18 if($patchedfv === "...") {  
19     @ob_end_clean(); die;  
20 }  
21 eval(base64_decode("JHVFUkN6ID0gJys9I  
FpYUy4uLj...ka0N4dE9KT2prcigpOyA="));  
22 @ini_restore('error_log');  
23 @ini_restore('display_errors');  
24 ...  
25
```

```
40 <?php  
41 error_reporting(0);  
42 @ini_set('error_log',NULL);  
43 @ini_set('log_errors',0);  
44 @ini_set('display_errors','Off');  
45 if(md5($_POST["pf"]) === "...")  
46     eval(base64_decode($_POST["..."]));  
47 ...  
48 if($patchedfv === "...") {  
49     @ob_end_clean(); die;  
50 }  
51 $uERCz = '+= ZXS...>68,Q';  
52 $kCxt0J0jkr = $uERCz(' ', '8ZfCK<.:>  
==72-XE08...RA715e<Ei>Z5M83fSbQ:0');  
53 $kCxt0J0jkr();  
54 @ini_restore('error_log');  
55 @ini_restore('display_errors');  
56 ...
```

(a) Normalized Program

(b) Deobfuscated Program 1

(c) Deobfuscated Program 2

**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

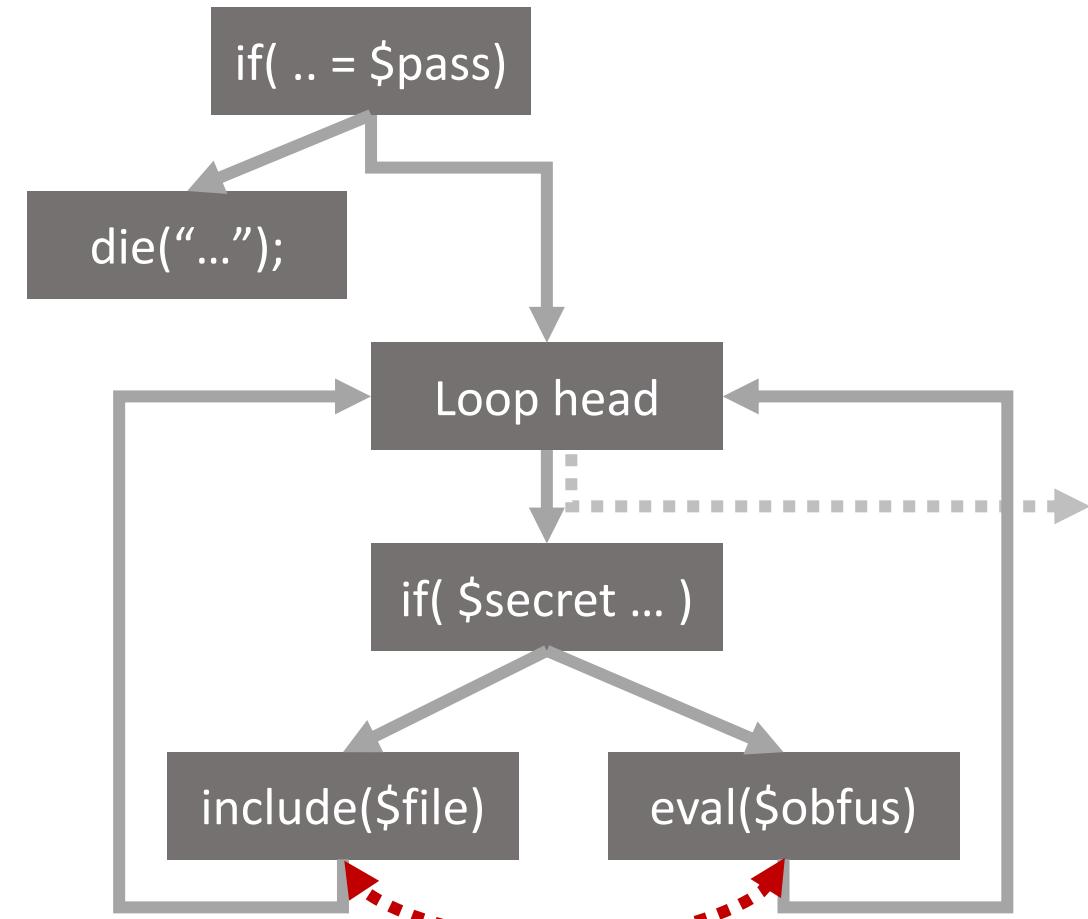
## 2. Multiple Layers of Obfuscation via Dynamic Constructs

- 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

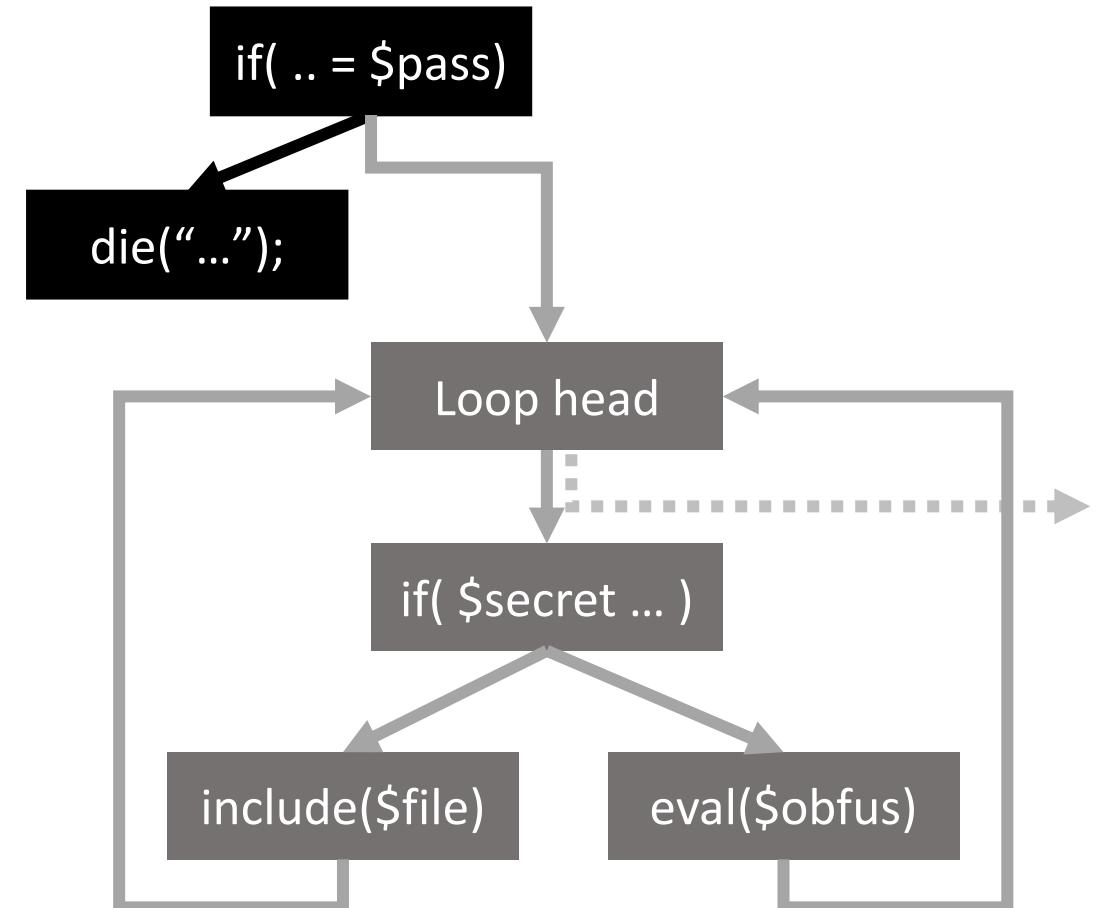
# Evasive and Multiple Layers of Obfuscation



```
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   }
}
```

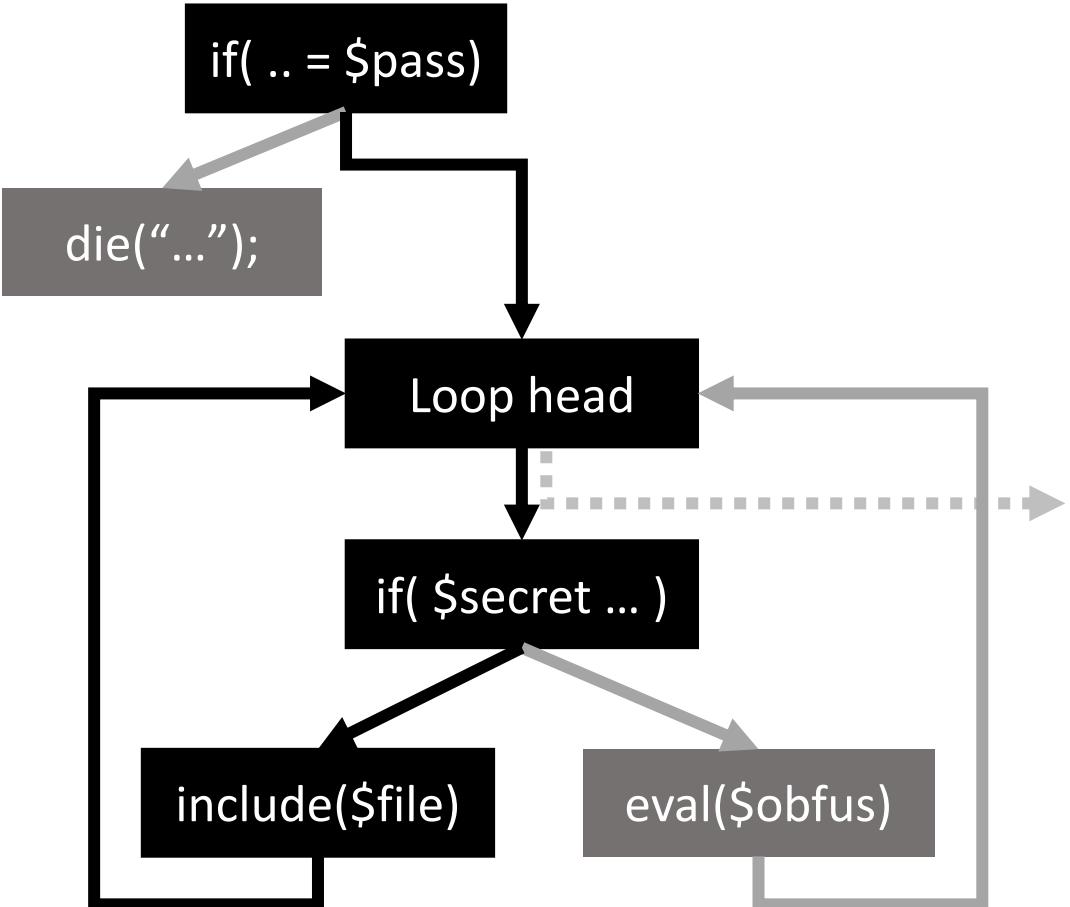
eval(\$obfuscated\_code) defines \$filename, and  
include(\$filename) will update \$obfuscated\_code

# 1. Evasive



```
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 }
```

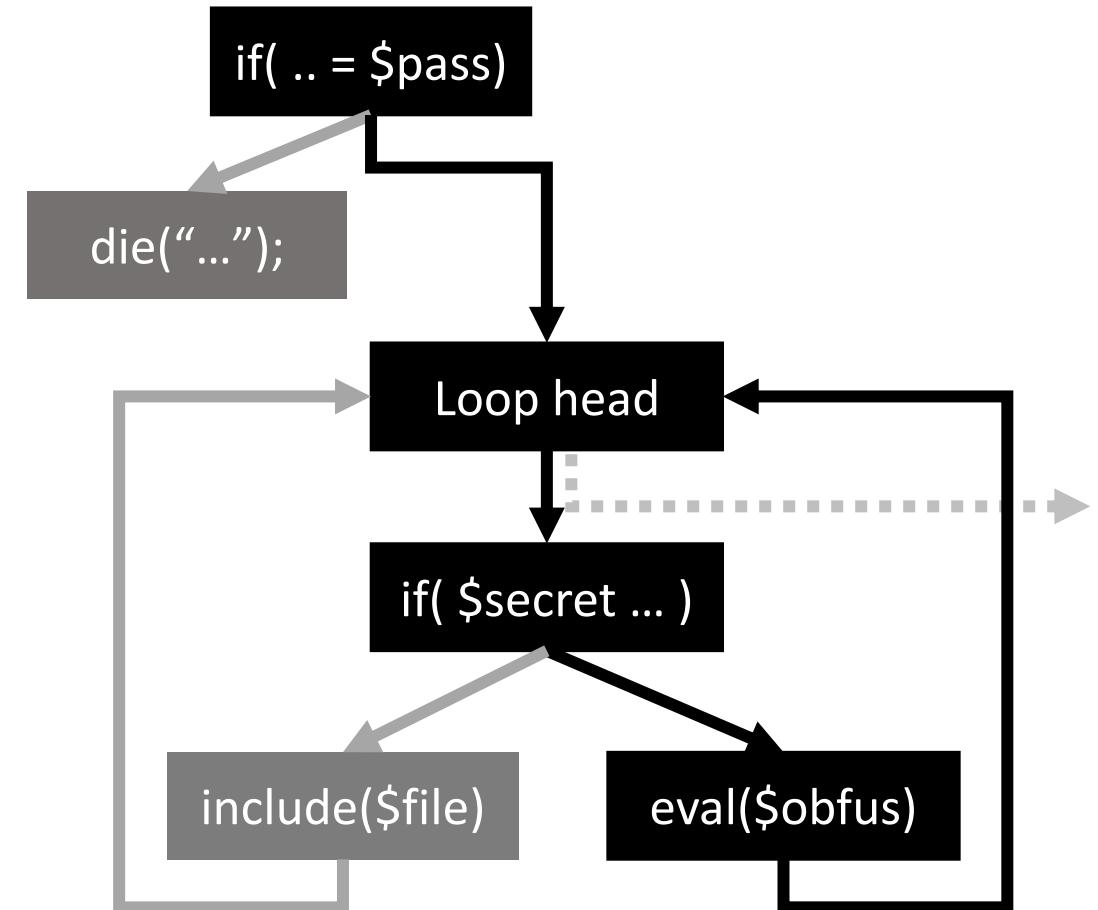
## 2. Multiple Layers of Obfuscation



```
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     }
}
```

No deobfuscation

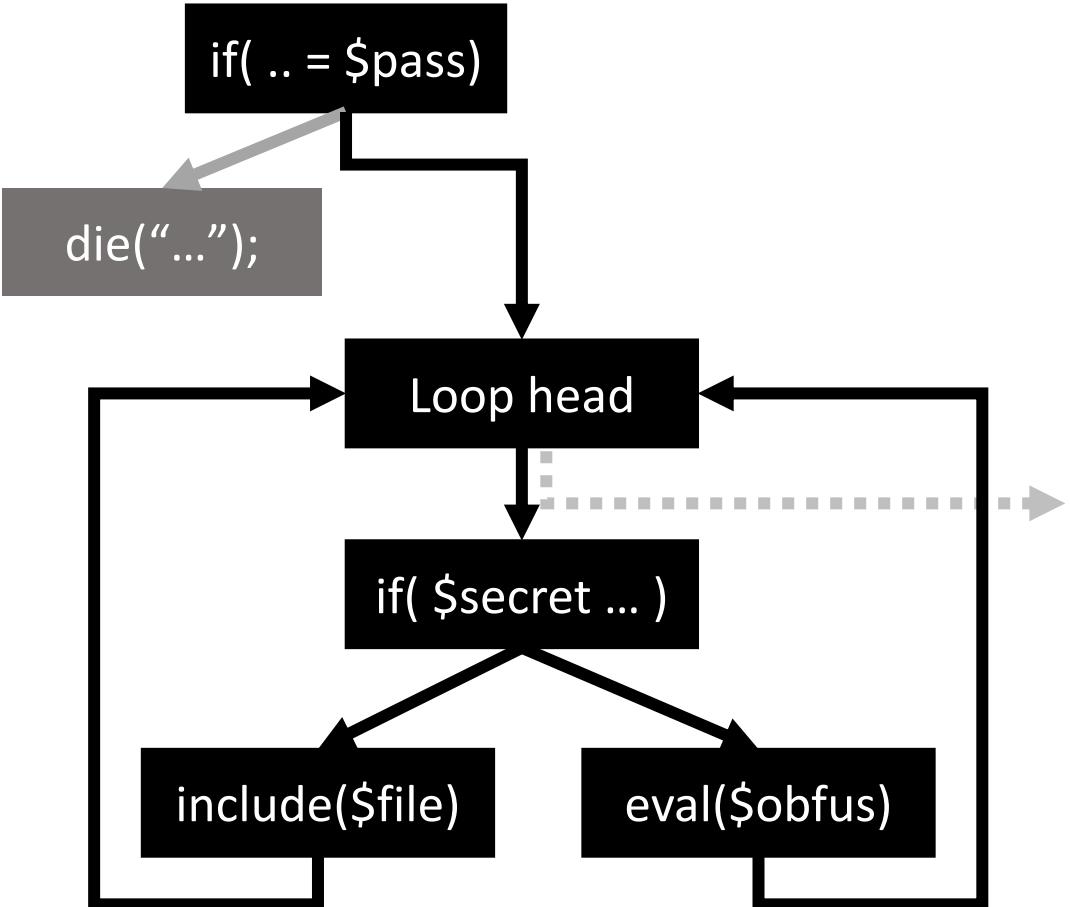
## 2. Multiple Layers of Obfuscation



Deobfuscation  
Layer 1

```
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 }
```

## 2. Multiple Layers of Obfuscation



```
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   }
}
```

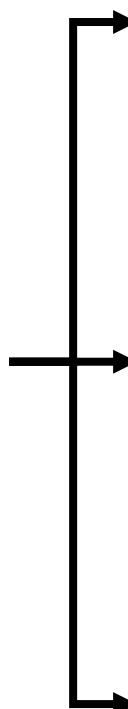
Deobfuscation  
Layer 2

# 3. Automated Malware Variant Generation

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

```
1 $s_pass = '4b34f78fb...';  
2 $x=gzinflate(base64_decode("7b1pe+041Sj80fM88x8  
U3XpT5chVXLV1LR1KFEVt1L5QnVw/3EmJm7hTufnvL0BSsm  
zLd1...+JDlu+vGCe/m0F3+e7PpQzuf97sMYN0MIA7DsAeZ  
PX/5/"));  
3 eval('?>'.$x);
```

(a) Original Malware



```
10 $s_pass = '4b34f78fb...';  
11 eval('$x=gzin'. 'flate(base'.'64_de'.'code("7b1p  
e+041Sj80fM88x8U3XpT5chVXLV1LR1KFEVt1L5QnVw/3Em  
Jm7hTufnvL0BSsmzLd1...+JDlu+vGCe/m0F3+e7PpQzuf9  
7sMYN0MIA7DsAeZPX/5/"));'  
12 eval('?>'.$x);
```

(b) Malware Variant 1

```
20 $s_pass = 'b4616d42a983401bcf344f9c18675777';  
21 eval('$x=gzi'. 'nflate(ba'.'se64_dec'.'ode("7b1p  
e+041Sj80fM88x8U3XpT5chVXLV1LR1KFEVt1L5QnVw/3Em  
Jm7hTufnvL0BSsmzLd1...+JDlu+vGCe/m0F3+e7PpQzuf9  
7sMYN0MIA7DsAeZPX/5/"));'  
22 eval('?>'.$x);
```

(c) Malware Variant 2

```
30 $s_pass = '62908bf72c21a3d8eaa23a55dec98e4b';  
31 eval('$x=g'.'zin'.'fla'.'te(base6'.'4_dec'.'ode  
("7b1pe+041Sj80fM88x8U3XpT5chVXLV1LR1KFEVt1L5Qn  
Vw/3EmJm7hTufnvL0BSsmzLd1...+JDlu+vGCe/m0F3+e7P  
pQzuf97sMYN0MIA7DsAeZPX/5/"));'  
32 eval('?>'.$x);
```

(d) Malware Variant 3

# 3. Automated Malware Variant Generation

- Changing \$s\_pass

```
1 $s_pass = '4b34f78fb...';  
2 $x=gzinflate(base64_decode("7b1pe+04lsj80fM88x8  
U3XpT5chVXLV1LR1KFEVt1L5QnVw/3EmJm7hTufnvL0BSsm  
zLd1...+JDlu+vGCe/m0F3+e7PpQzuf97sMYN0MIA7DsAeZ  
PX/5/"));  
3 eval('?>'.$x);
```

(a) Original Malware

```
10 $s_pass = '4b34f78fb...';  
11 eval('$x=gzin'. 'flate(base'.'64_de'.'code("7b1p  
e+04lsj80fM88x8U3XpT5chVXLV1LR1KFEVt1L5QnVw/3Em  
Jm7hTufnvL0BSsmzLd1...+JDlu+vGCe/m0F3+e7PpQzuf9  
7sMYN0MIA7DsAeZPX/5/"));'  
12 eval('?>'.$x);
```

(b) Malware Variant 1

```
20 $s_pass = 'b4616d42a983401bcf344f9c18675777';  
21 eval('$x=gzi'. 'inflate(ba'.'se64_dec'.'ode("7b1p  
e+04lsj80fM88x8U3XpT5chVXLV1LR1KFEVt1L5QnVw/3Em  
Jm7hTufnvL0BSsmzLd1...+JDlu+vGCe/m0F3+e7PpQzuf9  
7sMYN0MIA7DsAeZPX/5/"));'  
22 eval('?>'.$x);
```

(c) Malware Variant 2

```
30 $s_pass = '62908bf72c21a3d8eaa23a55dec98e4b';  
31 eval('$x=gz'. 'zin'. 'fla'. 'te(base6'. '4_dec'. 'ode  
("7b1pe+04lsj80fM88x8U3XpT5chVXLV1LR1KFEVt1L5Qn  
Vw/3EmJm7hTufnvL0BSsmzLd1...+JDlu+vGCe/m0F3+e7P  
pQzuf97sMYN0MIA7DsAeZPX/5/"));'  
32 eval('?>'.$x);
```

(d) Malware Variant 3

### 3. Automated Malware Variant Generation

- “\$x = gzinflate(base64\_decode” → “eval(‘\$x=gzip’.’flate...”

```
1 $s_pass = '4b34f78fb...';  
2 $x=gzinflate(base64_decode("7b1pe+041Sj80fM88x8  
U3XpT5chVXLV1LRlKFEVt1L5QnVw/3EmJm7hTufnvL0BSsm  
zLd1...+JDlu+vGCe/m0F3+e7PpQzuf97sMYN0MIA7DsAeZ  
PX/5/"));  
3 eval('?>'.$x);
```

(a) Original Malware

```
10 $s_pass = '4b34f78fb...';  
11 eval('$x=gzin'. 'flate(base'.'64_de'.'code("7b1p  
e+041Sj80fM88x8U3XpT5chVXLV1LRlKFEVt1L5QnVw/3Em  
Jm7hTufnvL0BSsmzLd1...+JDlu+vGCe/m0F3+e7PpQzuf9  
7sMYN0MIA7DsAeZPX/5/"));'  
12 eval('?>'.$x);
```

(b) Malware Variant 1

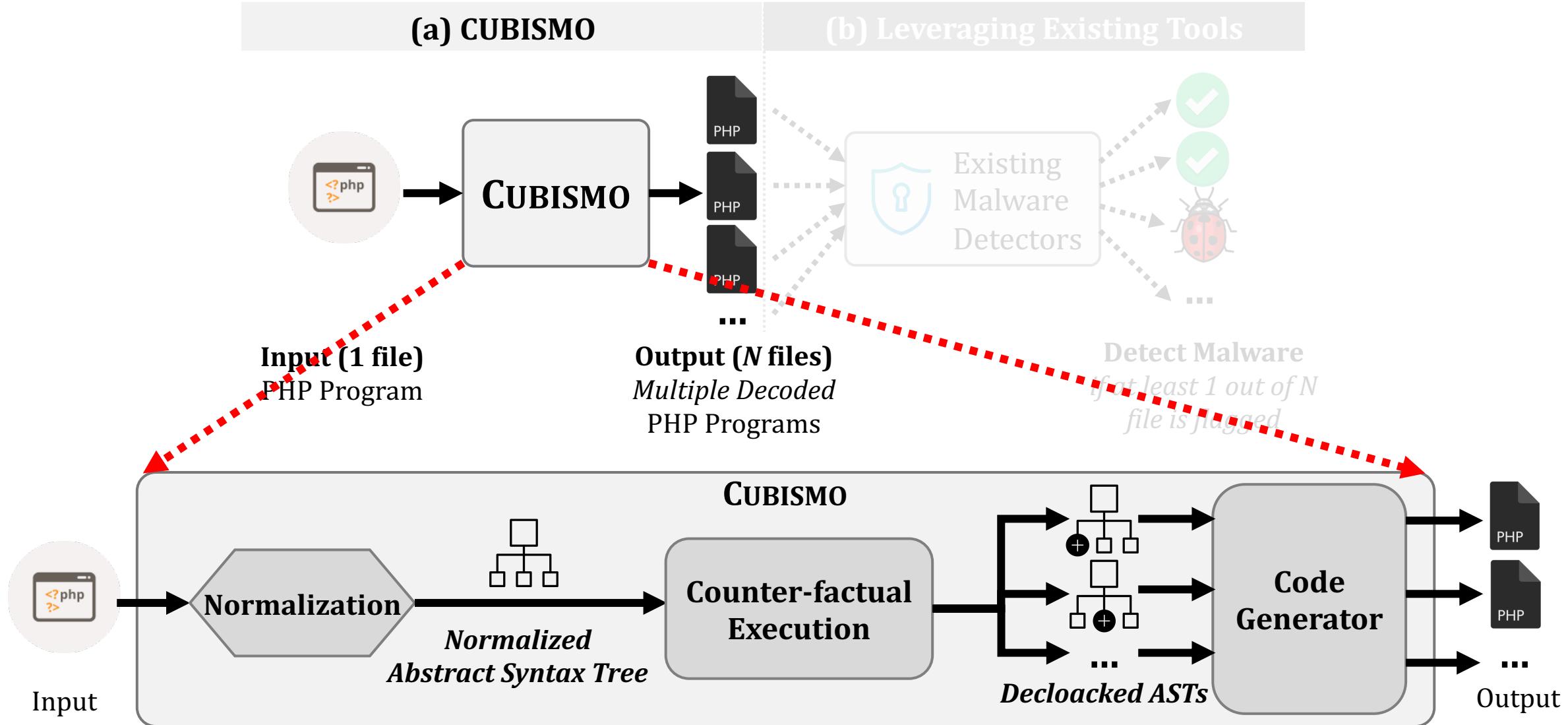
```
20 $s_pass = 'b4c16d42a983401bcf344f9c18675777';  
21 eval('$x=gzi'. 'inflate(ba'.'se64_dec'.'ode("7b1p  
e+041Sj80fM88x8U3XpT5chVXLV1LRlKFEVt1L5QnVw/3Em  
Jm7hTufnvL0BSsmzLd1...+JDlu+vGCe/m0F3+e7PpQzuf9  
7sMYN0MIA7DsAeZPX/5/"));'  
22 eval('?>'.$x);
```

(c) Malware Variant 2

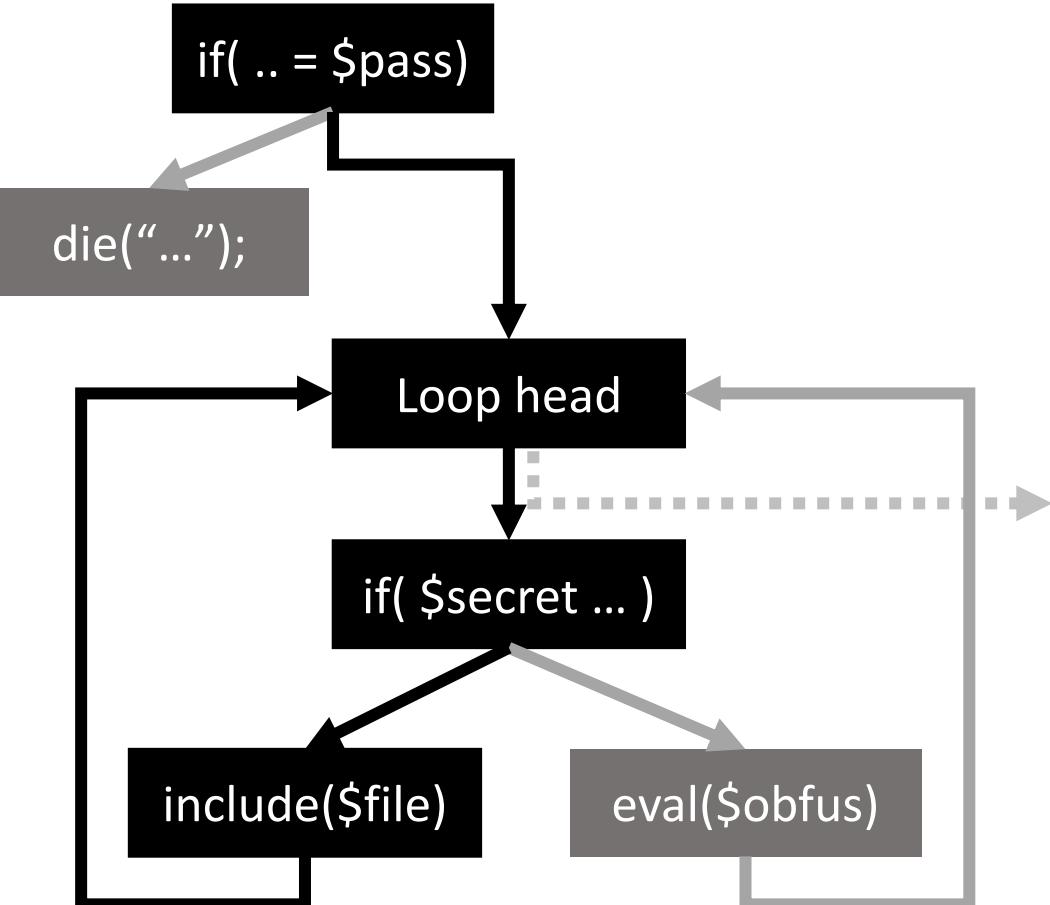
```
30 $s_pass = '62908bf72c21a3d8eaa23a55dec98e4b';  
31 eval('$x=g'. 'zin'. 'fla'. 'te(base6'. '4_dec'.'ode  
("7b1pe+041Sj80fM88x8U3XpT5chVXLV1LRlKFEVt1L5Qn  
Vw/3EmJm7hTufnvL0BSsmzLd1...+JDlu+vGCe/m0F3+e7P  
pQzuf97sMYN0MIA7DsAeZPX/5/"));'  
32 eval('?>'.$x);
```

(d) Malware Variant 3

# Overview: CUBISMO

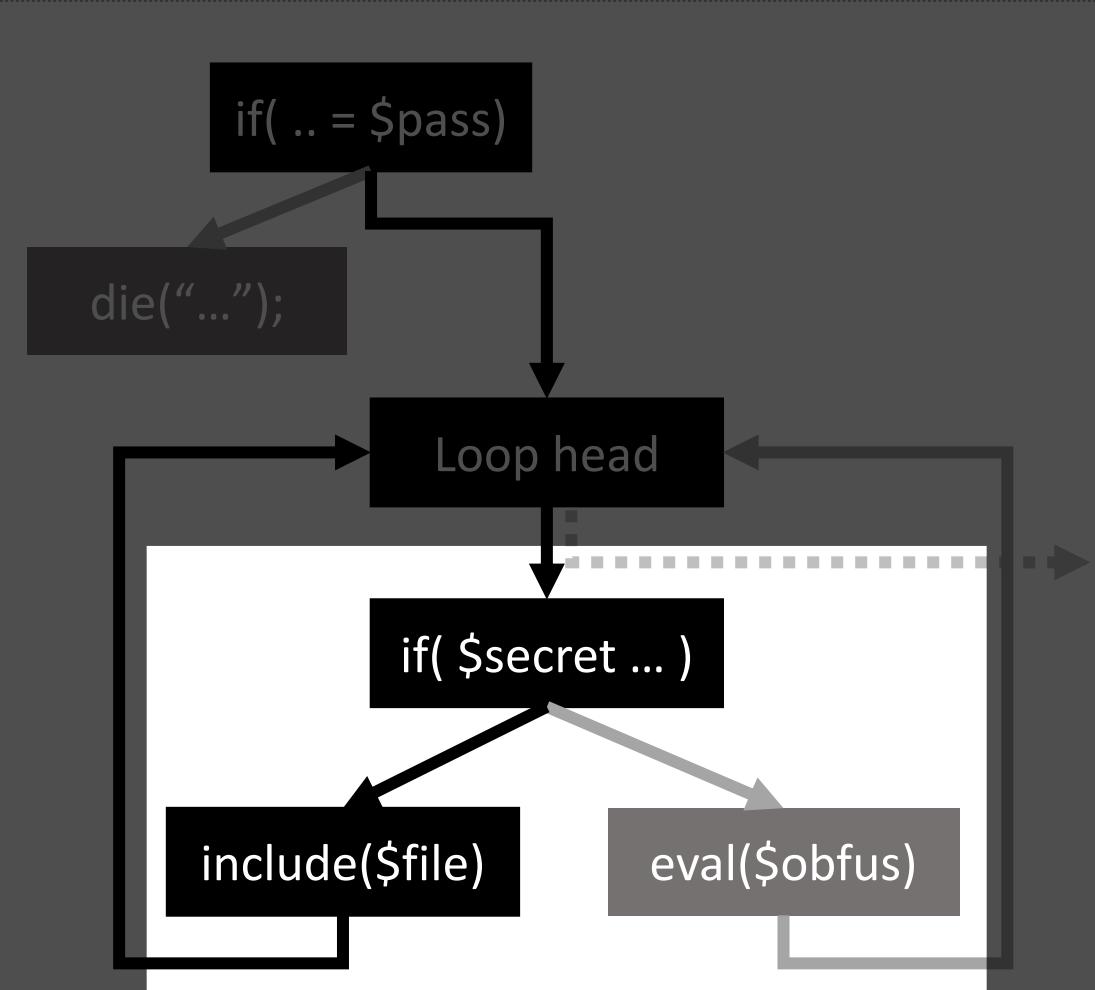


# Exposing Multiple Aspects of Malware



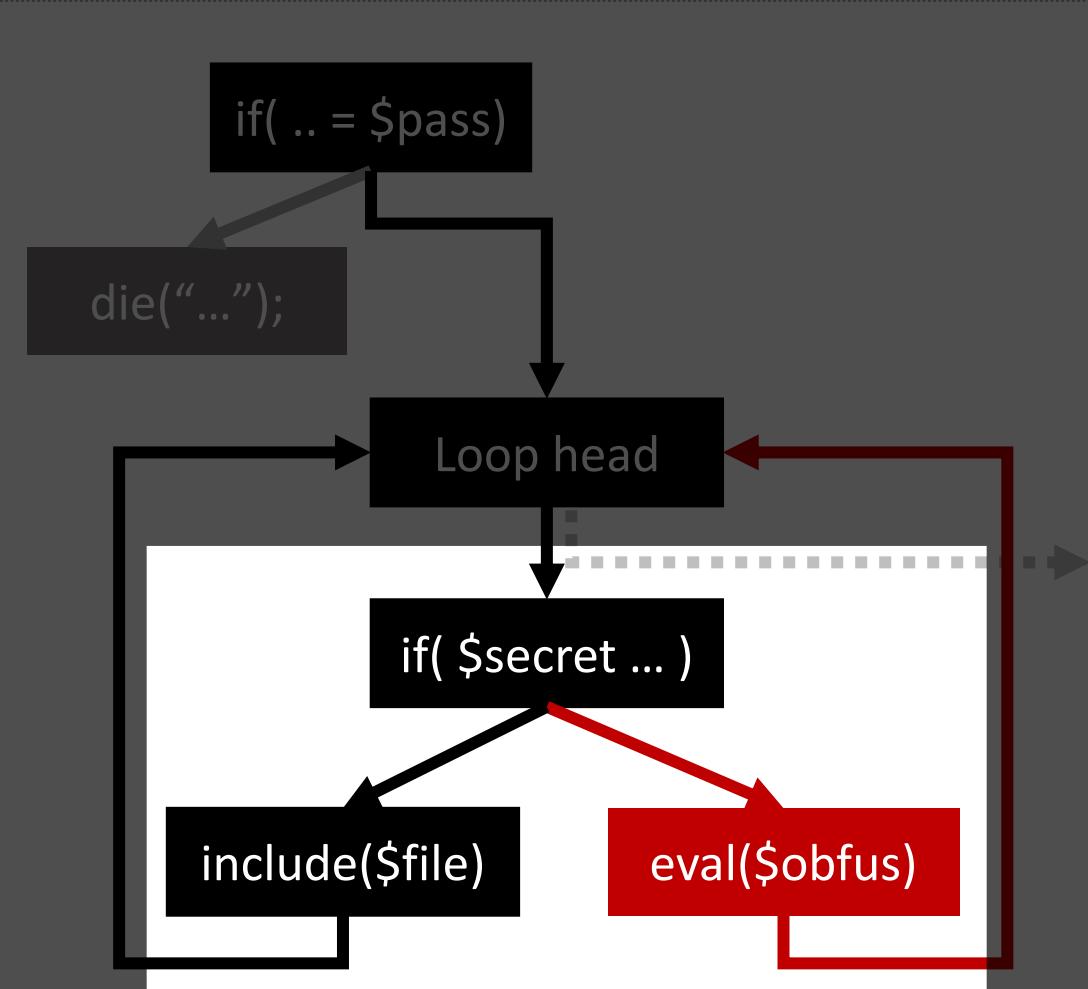
```
1 if ($_GET[1]!=$password)
2   die("Nothing to see here.");
3 for (...)
4   if ($secret === "...") {
5     include($filename);
6   } else {
7     eval($obfuscated_code);
8 }
```

# Counter-factual Execution [MalMax, CCS'19]



```
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 }
```

# Counter-factual Execution [MalMax, CCS'19]



```
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 }
```

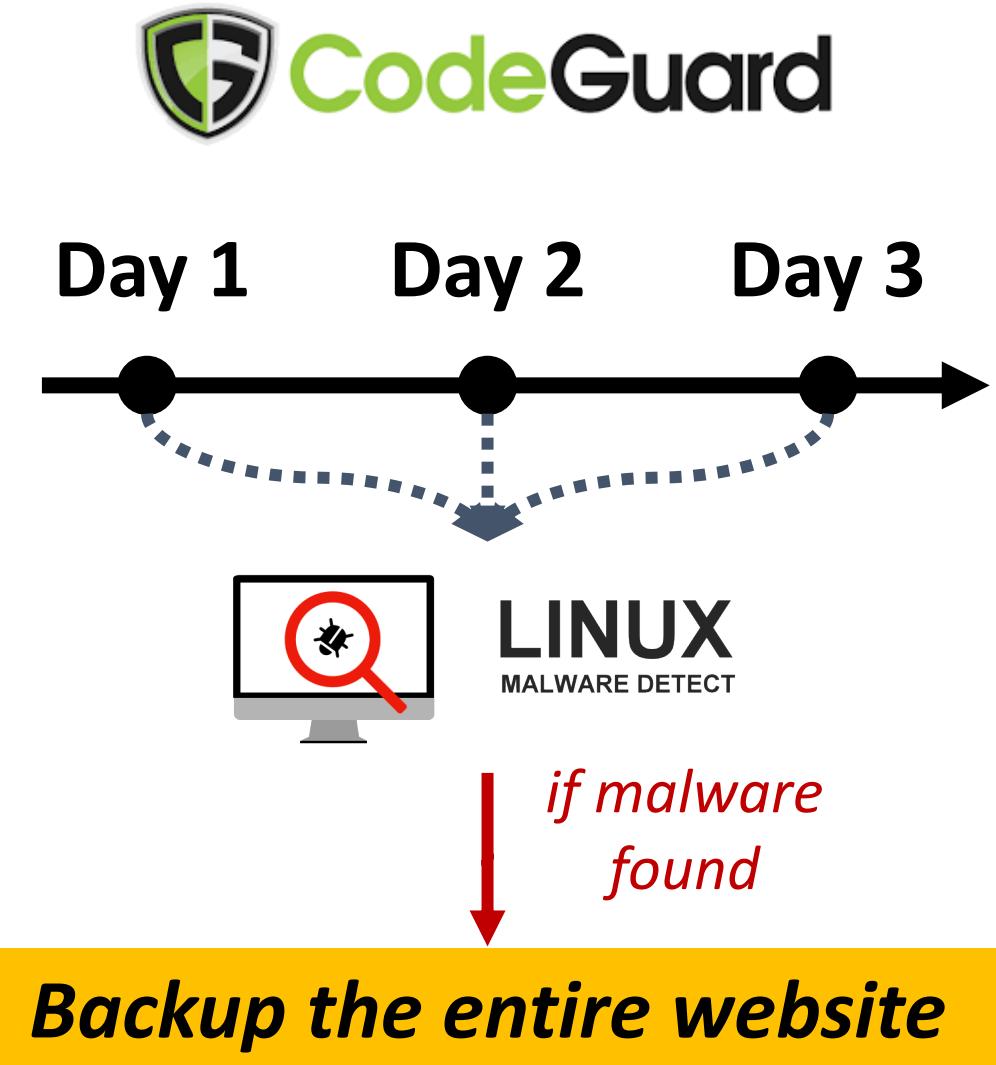
# More details in the paper

---

- **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.



# 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**

**700K files**

**1,269**

**352**

**56**

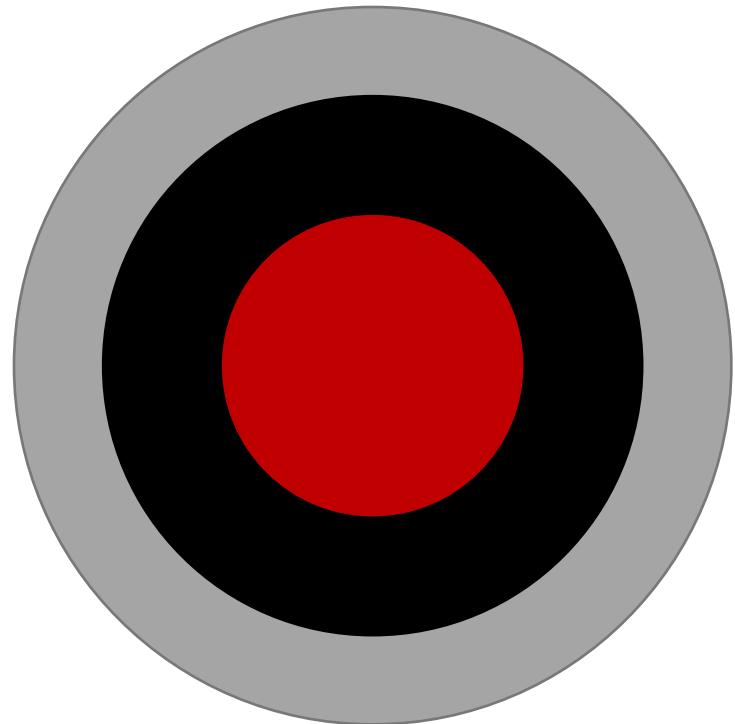
# Evaluation: Methodology

---

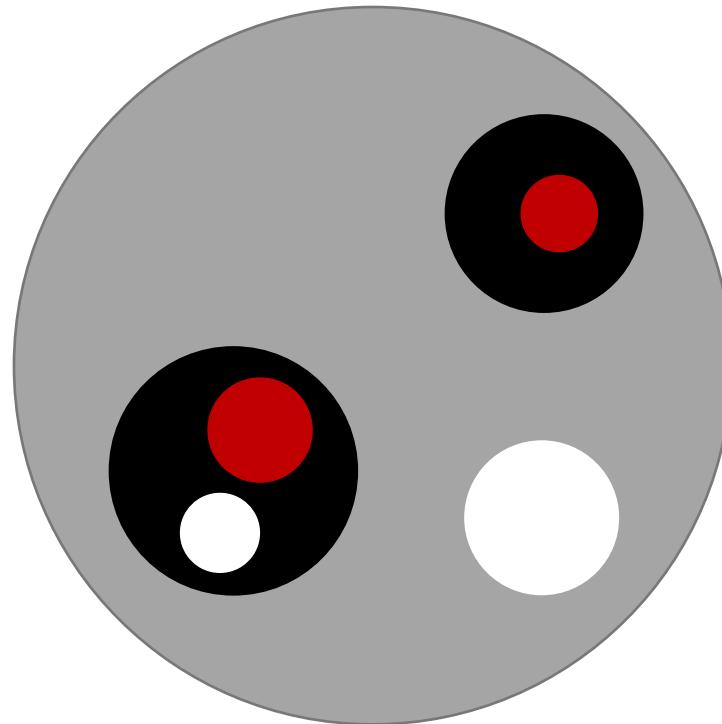
- We use **VirusTotal** (as an existing tool in our pipeline)
  - We feed malware to CUBISMO that produces multiple decloaked files
  - (a) We feed the decloaked files to VT
  - (b) We also feed the original file to VT and then we compare (a) and (b)
- **VirusTotal learns!** and we consider that
  -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

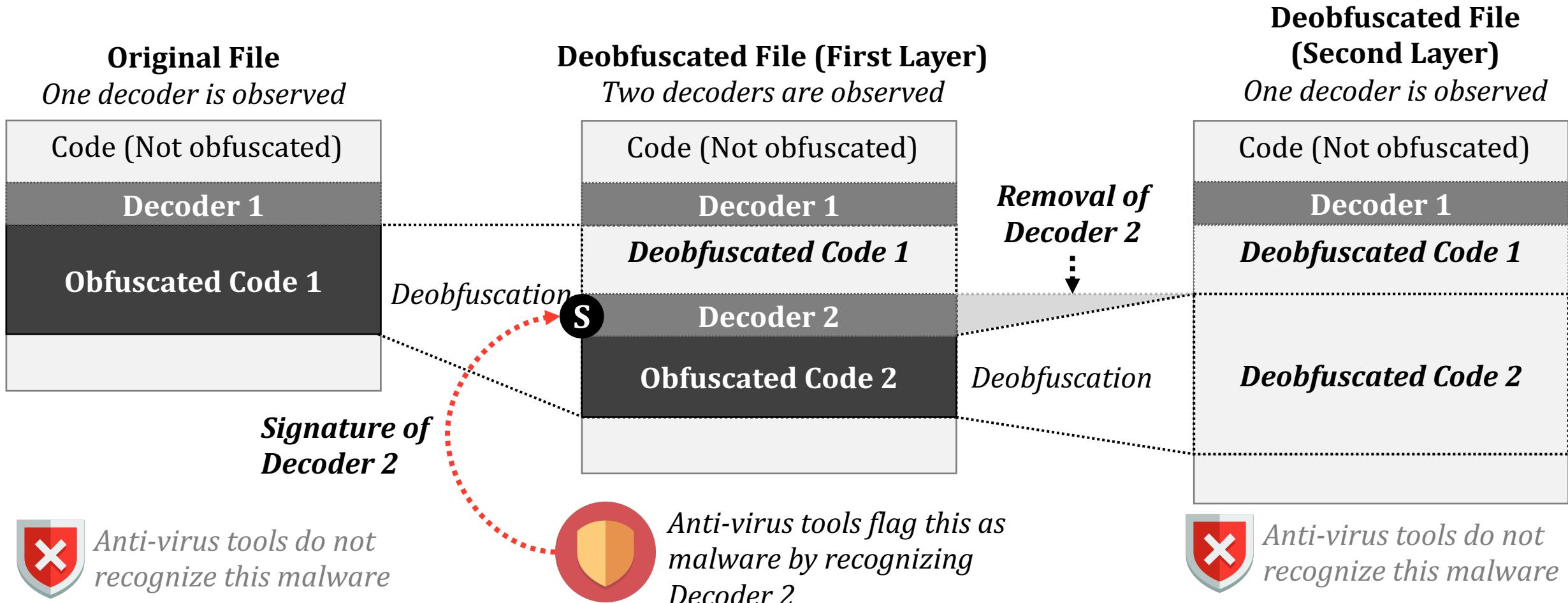


Naive Obfuscation



Advanced Obfuscation

# Evaluation: Every Layer Matters



# Evaluation: Everything Matters

	Orig.	Norm.	Layer 1	Layer 2	Layer 3	Layer 4		Orig.	Norm.	Layer 1	Layer 2	Layer 3	Layer 4	
m1	0	1	2	-	-	-		m29	0	0	1	1	1	-
m2	0	0	1	1	1	2		m30	0	0	1	-	-	-
m3	0	0	1	-	-	-		m31	0	0	1	-	-	-
m4	0	0	1	-	-	-		m32	0	0	1	-	-	-
m5	0	0	1	1	-	-		m33	0	1	3	1	-	-
m6	0	0	1	-	-	-		m34	0	0	1	1	0	1
m7	0	0	1	1	-	-		m35	0	0	1	-	-	-
m8	0	0	1	-	-	-		m36	0	1	1	-	-	-
m9	0	0	1	-	-	-		m37	0	0	1	-	-	-
m10	0	1	1	-	-	-		m38	0	1	1	-	-	-
m11	0	0	3	1	-	-		m39	0	0	1	-	-	-
m12	0	0	1	-	-	-		m40	0	0	3	1	3	3
m13	0	0	1	-	-	-		m41	0	1	1	-	-	-
m14	0	1	3	1	-	-		m42	0	1	1	-	-	-
m15	0	0	1	-	-	-		m43	0	0	1	1	1	-
m16	0	0	1	1	-	-		m44	0	0	1	1	-	-
m17	0	0	0	0	-	-		m45	0	0	1	1	1	1
m18	0	0	1	-	-	-		m46	0	0	1	-	-	-
m19	0	1	1	-	-	-		m47	0	0	1	-	-	-
m20	0	0	1	1	1	-		m48	0	0	1	-	-	-
m21	0	0	1	5	-	-		m49	0	0	1	1	-	-
m22	0	0	2	2	-	-		m50	0	0	0	-	-	-
m23	0	0	1	-	-	-		m51	0	0	1	-	-	-
m24	0	0	1	-	-	-		m52	0	0	1	-	-	-
m25	0	0	3	-	-	-		m53	0	1	2	1	-	-
m26	0	0	1	1	-	-		m54	0	0	1	-	-	-
m27	0	0	0	0	-	-		m55	0	0	1	-	-	-
m28	0	0	1	-	-	-		m56	0	0	1	-	-	-

# 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.

```
1 <?php
2 namespace A {
3     class ClassA {
4         function funcA() {
5             $f = basename(__FILE__, '.php');
6             extract_malicious_code( $f );
7         }
8     }
9 }
10 namespace {
11     $a = new \A\ClassA();
12     ...
13 }
14 // Comment including malicious code
15 // Comment to break parser
```

Normalization

(a) Original Program  
*(PHP-Parser Crash)*

```
1 <?php
2 namespace A {
3     class ClassA {
4         function funcA() {
5             $f = basename(__FILE__, '.php');
6             extract_malicious_code( $f );
7         }
8     }
9 }
10 namespace {
11     $a = new \A\ClassA();
12     ...
13 }
14
```



(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

