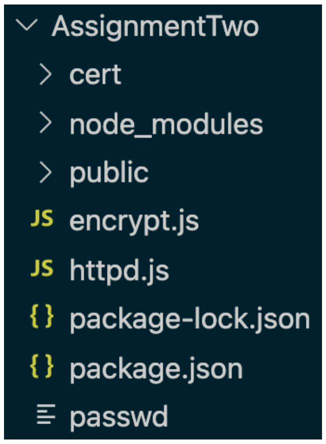
Application Overview

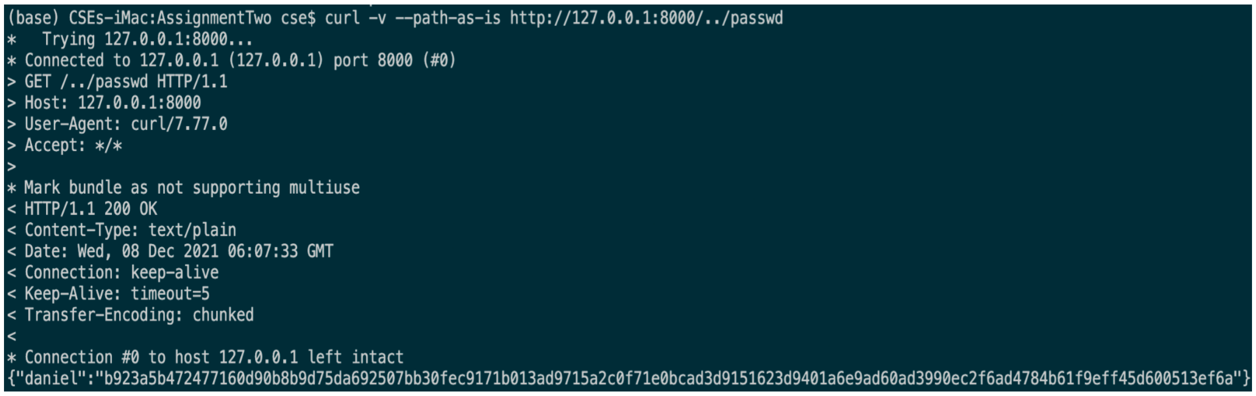
Our application is built for the purpose of replicating an app for a smart home system. The most challenging part while building the application was to change the values of buttons simultaneously on multiple instances of the web application if only one instance is changed. In this way, the logical integrity of the application was preserved. For the authentication part, the username and password of the users were stored in a file ‘passwd’ which did not have any extension and was kept outside of the public directory for safety. Still, it was vulnerable to Path Traversal attack. The folder structure is given below.



```
AssignmentTwo
  cert
  node_modules
  public
  encrypt.js
  httpd.js
  package-lock.json
  package.json
  passwd
```

Stealing the Password file

Using CURL it was possible to access the contents of the ‘passwd’ file. The command along with outputs are shown below.



```
(base) CSEs-iMac:AssignmentTwo cse$ curl -v --path-as-is http://127.0.0.1:8000/../passwd
*   Trying 127.0.0.1:8000...
* Connected to 127.0.0.1 (127.0.0.1) port 8000 (#0)
> GET /../passwd HTTP/1.1
> Host: 127.0.0.1:8000
> User-Agent: curl/7.77.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< Content-Type: text/plain
< Date: Wed, 08 Dec 2021 06:07:33 GMT
< Connection: keep-alive
< Keep-Alive: timeout=5
< Transfer-Encoding: chunked
<
* Connection #0 to host 127.0.0.1 left intact
{"daniel":"b923a5b472477160d90b8b9d75da692507bb30fec9171b013ad9715a2c0f71e0bcad3d9151623d9401a6e9ad60ad3990ec2f6ad4784b61f9eff45d600513ef6a"}
```

Preventing the attack

Encrypting the ‘passwd’ file

If the contents of the file are encrypted then it will be very less helpful to the eavesdropper even though s/he unethically gets access to the file. I created a separate **‘encrypt.js’** file which will encrypt the passwords but not the usernames. The code snippet is given below.

```
const crypto = require('crypto')
const fs = require('fs')
credentials = fs.readFileSync('passwd',{encoding:"utf-8"})
credentials = (JSON.parse(String(credentials)))

for(key in credentials)
{
   derivedKey = crypto.pbkdf2Sync(credentials[key], 'salt', 100000, 64, 'sha512');
   derivedKey = derivedKey.toString('hex')
   credentials[key] = derivedKey
}

fs.writeFileSync('passwd', JSON.stringify(credentials),{encoding:"utf-8"})
```

File before encrypting
{"daniel":"fisksoppa"}
File after encrypting
{"daniel":"b923a5b472477160d90b8b9d75da692507bb30fec9171b013ad9715a2c0f71e0bcad3d
9151623d9401a6e9ad60ad3990ec2f6ad4784b61f9eff45d600513ef6a"}

Checking file extension name before giving access to the file

If the user wants to access any file without an extension then it will be blocked using the following condition.

```
if (extname != ".ico" && extname != "") {
   filePath = "public" + req.url;
   content = fs.readFileSync(filePath, { encoding: "utf-8" });
   contentType = mimeTypes[extname];
   sendResponse(res);
 }
```