

# 天价手续费

核心问题：intToBuffer不支持传入浮点型的数据

分析：

主要讨论的代码问题是maxPriorityFeePerGas和maxFeePreGas这两个参数的值，这两个参数都是由toBuffer处理的

```
1 this.maxFeePerGas = new BN(toBuffer(maxFeePerGas === '' ? '0x' : maxFeePerGas))
2   this.maxPriorityFeePerGas = new BN(
3     toBuffer(maxPriorityFeePerGas === '' ? '0x' : maxPriorityFeePerGas)
4   )
```

toBuffer调用了intToBuffer函数，将int转成了Hex，判断是否被2整除，若不行则开头添加“0”，

```
function intToBuffer(i) {
  var hex = intToHex(i);
  return new Buffer(padToEven(hex.slice(2)), 'hex');
}
```

```
function intToHex(i) {
  var hex = i.toString(16); // eslint-disable-line
  return '0x' + hex;
}
```

```
function padToEven(value) {
  var a = value; // eslint-disable-line

  if (typeof a !== 'string') {
    throw new Error('[ethjs-util] while padding to even, value must be string, is currently ' + typeof a);
  }
  if (a.length % 2) {
    a = '0' + a;
  }
  return a;
}
```

以出错实例 33974229950.550003 来看，经过intToHex和padToEven处理后得到 7e9059bbe.8ccd，使用浏览器js处理分析，处理转换部分得到的结果 7e9059bbe.8ccd 填充到buffer中时，内容为 126, 144, 89, 187, 14, 140, 205，对应 7e, 90, 59, bb, e, 8c, cd，e后面的小数点不见了。因为在hexWrite这个函数使用了parseInt对切分后的数据进行解析，

```


858 function hexWrite (buf, string, offset, length) {
859   offset = Number(offset) || 0
860   var remaining = buf.length - offset
861   if (!length) {
862     length = remaining
863   } else {
864     length = Number(length)
865     if (length > remaining) {
866       length = remaining
867     }
868   }
869
870   // must be an even number of digits
871   var strLen = string.length
872   if (strLen % 2 !== 0) throw new TypeError('Invalid hex string')
873
874   if (length > strLen / 2) {
875     length = strLen / 2
876   }
877   for (var i = 0; i < length; ++i) {
878     var parsed = parseInt(string.substr(i * 2, 2), 16)
879     if (isNaN(parsed)) return 1
880     buf[offset + i] = parsed
881   }
882   return i
883 }

```

`parseInt('e.',16) -> 14` `===` `parseInt('e',16) -> 14`，小数点被`parseInt`忽略掉了，导致最终写入buffer中的数据为 `7e9059bbe8ccd`

## parseInt

`parseInt(string, radix)` 解析一个字符串并返回指定基数的十进制整数，`radix` 是2-36之间的整数，表示被解析字符串的基数。

 JavaScript Demo: Standard built-in objects - parseInt()

```

1 console.log(parseInt('e', 16));
2 console.log(parseInt('e.', 16));
3
4
5

```

Run › > 14

Reset > 14

为何现在才出现问题：只有资金量非常大的钱包才会受到影响，其他用户看到的都会是交易失败

## SuShiSwap供应链投毒

由于SuShi的GitHub相关存储库删除，无法看到恶意代码

攻击者访问SushiSwap的github存储库，正常来说没有权限的人访问会出现404，应该是SushiSwap缺少某些安全程序，让攻击者可以避开访问权限，成功将恶意代码推送到“miso-studio”存储库，并向Miso的平台前端注入了恶意代码，将合约地址改为攻击者的私有地址获利。

（攻击者干扰或劫持软件流程，插入恶意代码，导致制成品的大量客户受影响受影响时就会发生软件供应链攻击。

当软件构件中使用的代码库或单个组件被污染，软件更新二进制文件被“木马化”，代码签名证书被盗，甚至当提供软件即服务的服务器被攻破，都可能发生这种情况，因此与孤立的安全漏洞相比，成功的供应链攻击会产生更广泛的影响和破坏。

)

尽管任何人都可以提议为公共 GitHub 存储库做出贡献，但只有部分个人可以访问或贡献私有存储库。即便如此，理想情况下，提交也应该得到项目中受信任成员的验证和批准。

“**吸血鬼攻击**”的创造者、加密货币爱好者 Martin Krung 想知道攻击者的拉取请求在被合并到代码库之前是否经过了适当的审查，他从贡献者那里得到了见解：

“

我见过 40 多个文件更改后立即获得批准的 PR。没有代码所有权。

— adamazad.eth [ (@adamzad) 2021 年 9 月 17 日

SushiSwap 编译的**粗略分析**（现已被 SushiSwap 删除，但**在此处备份**）试图追踪攻击者并引用多个数字身份。SushiSwap 认为 GitHub 用户 AristoK3 与 Twitter 句柄 eratos1122 相关联，尽管后者的回应尚无定论。“这真的很疯狂……请删除它并向所有人说‘对不起’……如果没有，我将分享我拥有的所有 MISO 项目 [原文如此]（你知道我在 MISO 项目上的工作很好）”，eratos1122**回应**道。

由于分析中提到的一些数字身份仍未得到验证，Ars 避免提及这些，直到有更多信息可用。我们已经联系了德隆和涉嫌攻击者以了解更多信息。我们正在等待他们的回应。