DAAC智能合约无符号整数溢出漏洞分析

DAAC

Distributed Adult Art & Culture（DAAC）旨在打造一个基于IPFS和区块链的下一代分布式文化娱乐影视平台。推动文化影视的发展，创造出更多的商业价值。通过提供去中心化的相关软件、DAPP、在线支付服务等内容，为文化影视产业内容存储、版权开发、数据下载、匿名支付等问题提供一个完整的解决方案。同时，基于标准型通证DAAC在VR影视、粉丝经济、5G直播、版权溯源等多个场景中的应用，不断完善和拓展文化影视行业的发展，彻底改变传统文化影视产业的现状。

- 总供应量 1,000,000,000 DAAC
- 上架交易所 3家
- 均价0.1670

影响范围

DAAC https://etherscan.io/address/0xA4112bE97aca5b0cAbf5e1EFB35C99A0459B30c2

漏洞简析

UTC时间Aug-04-2019 10:20:41 AM +UTC，DAAC智能合约遭受黑客攻击，共计转走 `115,792,089,237,316,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000.584007913129639935` 数量代币。

根据交易hash `https://etherscan.io/tx/0xa11ad2e8575e006f2a8e14bc77e03b482ee2b49228a97dcad95e2c8b42b44ce8` 查询到数据输入，确定受攻击的函数为 `batchTransfers(address[] receivers, uint256[] amounts)`

攻击者输入的参数为:

```
receivers   address[]   5bdd5dae9282700c88f3437df60a8a8917134ff2, 1a2228110522c194eaaf2ab215789f8b5f093740
amounts uint256[]   115792089237316195423570985008687907853269984665640564039457584007913129639935, 2
```

数据输入:

#	Name	Type	Data
0	receivers	address[]	5bdd5dae9282700c88f3437df60a8a8917134ff2 1a2228110522c194eaaf2ab215789f8b5f093740
1	amounts	uint256[]	115792089237316195423570985008687907853269984665640564039457584007913129639935 2

Decoded input inspired by Canoe Solidity

切换回来

其中 `115792089237316195423570985008687907853269984665640564039457584007913129639935` 为 `uint256 - 1`

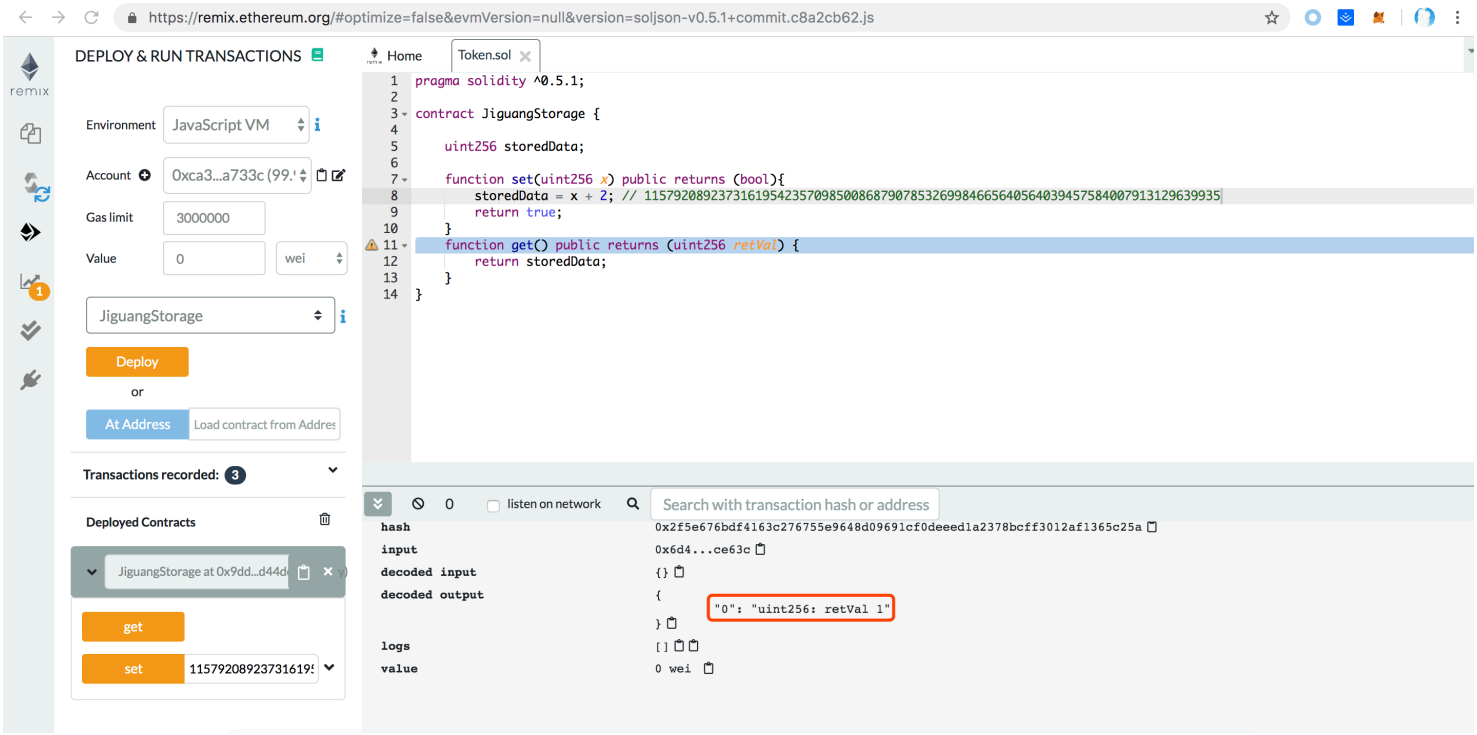
通过对函数 `batchTransfers` 分析可知，该函数实现批量转账功能，通过对输入的数组账户地址和数组代币量进行批量转账。在第535行对数组总代币数量进行叠加计算， `注意这里直接把输入的代币数量进行了算术运算，并未使用safeMath方法` 。通过上面攻击者的输入参数 `115792089237316195423570985008687907853269984665640564039457584007913129639935, 2` 可知，在for循环的地539行发生了无符号算术溢出，
即 `uint256(115792089237316195423570985008687907853269984665640564039457584007913129639935) + uint256(2) = uint256(1)` ，所以变量 `totalAmount` 的值经过for循环运算后变为 `1` 。

```
526     //address1 balances 248
527     function batchTransfers (address[] memory receivers, uint256[] memory amounts) public whenRunning returns (bool) {
528             uint receiveLength = receivers.length; // address1, address2
529             require(receiveLength == amounts.length);// [uint256 -1 , uint256(2)] = 1
530
531     uint receiverCount = 0;
532     uint256 totalAmount = 0;
533     uint i;
534     address r;
535     for (i = 0; i < receiveLength; i ++) {
536             r = receivers[i];
537             if (r == address(0) || r == owner) continue;
538             receiverCount ++;
539             totalAmount += amounts[i]; // 1
540     }
541     require(totalAmount > 0); // 1
542     require(canPay(msg.sender, totalAmount)); //1
543
544     wallets[msg.sender] -= totalAmount;// 24799999
545     uint256 amount;
546     for (i = 0; i < receiveLength; i++) {
547             r = receivers[i];
548             if (r == address(0) || r == owner) continue;
549             amount = amounts[i];
550             if (amount == 0) continue;
551             wallets[r] = wallets[r].add(amount);
552             emit Transfer(msg.sender, r, amount);
553     }
554     return true;
555     }
```

关于无符号整数回绕问题，这里以Go语言为例子，其他静态语言如c/c++(solidity用c++实现)，拥有同样的问题。

```
Go中存在无符号整数使用不当所导致的问题。下表中列出ANSI标准定义的无符号整数类型及范围
类型   位数   取值范围
uint8   8       0 到 255
uint16  16  0 到 65535
uint32  32      0 到 4294967295
uint64  64      0 到 18446744073709552000
涉及无符号整数的计算当数值超过无符号整数的取值范围时会发生回绕。如：无符号整数的最大值加1会返回0，而无符号整数最小值减1则会返回该类型的最大值。造成无符号整数运算回绕的操作符有“+”、“-”、“*”、“++”、“—”、“+=”、“-=”、“*=”、“<<=”、“<<”等。
```

编写测试智能合约无符号整数溢出测试代码，通过传入参数 `115792089237316195423570985008687907853269984665640564039457584007913129639935` 参与运算后可知，参数storedData成功溢出为 `1` ,这也是DAAC智能合约函数 `batchTransfers` 中攻击者输入的参数。



回到 `batchTransfers` 函数，经过for运算后， `totalAmount` 成功溢出为 `1` ,通过了第541，542行校验，在第544行进行 `wallets[msg.sender] -= totalAmount;` ，实际上是 `wallets[msg.sender] -= 1;` ,即攻击者的代币数量只减少 `1` 。然后通过第551行对地址 `r` 的代币数量进行增加，增加的量为 `amounts[0]` ，通过前面的分析，可知攻击者传入的 `amounts` 参数为 `115792089237316195423570985008687907853269984665640564039457584007913129639935, 2` ，即 `amounts[0] == 115792089237316195423570985008687907853269984665640564039457584007913129639935` ，所以这里是造成大量代币转移的主要原因。

```
526     //address1 balances 248
527     function batchTransfers (address[] memory receivers, uint256[] memory amounts) public whenRunning returns (bool) {
528             uint receiveLength = receivers.length; // address1, address2
529             require(receiveLength == amounts.length);// [uint256 -1 , uint256(2)] = 1
530
531     uint receiverCount = 0;
532     uint256 totalAmount = 0;
533     uint i;
534     address r;
535     for (i = 0; i < receiveLength; i ++) {
536             r = receivers[i];
537             if (r == address(0) || r == owner) continue;
538             receiverCount ++;
539             totalAmount += amounts[i]; // 1
540     }
541     require(totalAmount > 0); // 1
542     require(canPay(msg.sender, totalAmount)); //1
543
544     wallets[msg.sender] -= totalAmount;// 24799999
545     uint256 amount;
546     for (i = 0; i < receiveLength; i++) {
547             r = receivers[i];
548             if (r == address(0) || r == owner) continue;
549             amount = amounts[i];
550             if (amount == 0) continue;
551             wallets[r] = wallets[r].add(amount);
552             emit Transfer(msg.sender, r, amount);
553     }
554     return true;
555     }
```

通过事件日志可知， `batchTransfers` 函数确实触发了两次 `Transfer` ，一次代币数量为 `115792089237316195423570985008687907853269984665640564039457584007913129639935` ，收件人为 `0x5bdd5dae9282700c88f3437df60a8a8917134ff2` 。

概况　事件日志 (2)　状况变化 新　评论

交易收据事件日志

73

地址 0xa4112be97aca5b0cabf5e1efb35c99a0459b30c2

名称 Transfer (index_topic_1 address from, index_topic_2 address to, uint256 value)

题目 0 0xddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f55a4df523b3ef

1 十六进制 → 0x000000000000000000000000f4c01cb969dbd5ca21dbd82016f289612d904731

2 十六进制 → 0x000000000000000000000000bdd5dae9282700c88f3437df60a8a8917134ff2

数据 Num → 1.157920892373161954235709850086879078532699846656405640394575840079131296399 35e+77

74

地址 0xa4112be97aca5b0cabf5e1efb35c99a0459b30c2

名称 Transfer (index_topic_1 address from, index_topic_2 address to, uint256 value)

题目 0 0xddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f55a4df523b3ef

1 十六进制 → 0x000000000000000000000000f4c01cb969dbd5ca21dbd82016f289612d904731

2 十六进制 → 0x000000000000000000000000a2228110522c194eaaf2ab215789f8b5f093740

数据 Num → 2

8. balanceOf

user (address)

0x5bdd5dae9282700c88f3437df60a8a8917134ff2

Query

uint256

[**balanceOf** method Response]

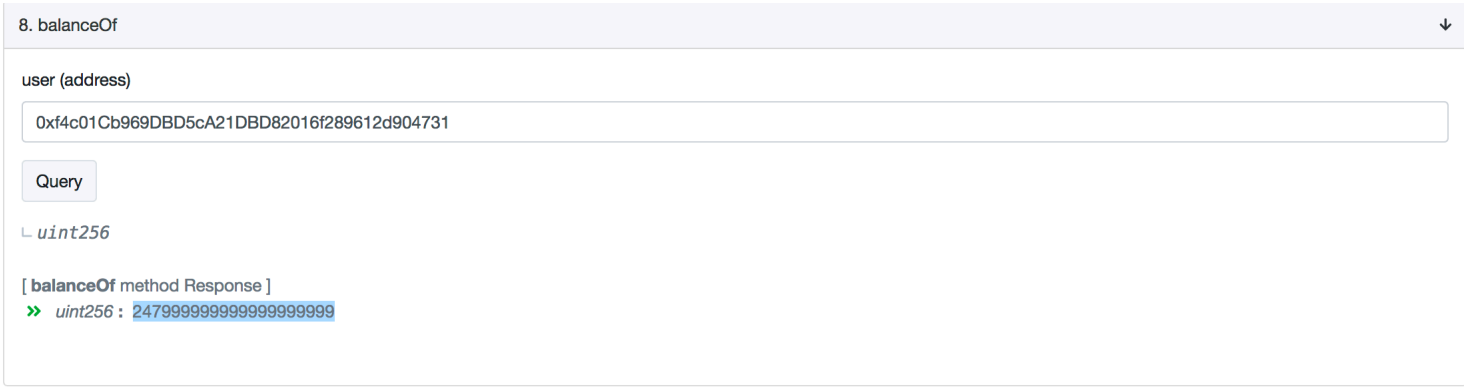
» *uint256* : 115792089237316195423570985008687907853269984665640514039457584007913129639935

通过对攻击者地址 `0xf4c01Cb969DBD5cA21DBD82016f289612d904731` 分析可知，攻击者先是向自己的地址转入 `248DAAC(小数18位)` ，然后攻击 `batchTransfers` 函数后，代币数量减少 `wallets[msg.sender] -= 1;` ,即攻击者的代币数量只减少 `1` ，为 `247999....(18位9)` 。这也验证了前面的分析。最终攻击者通过漏洞函数 `batchTransfers` 向地址 `0x5bdd5dae9282700c88f3437df60a8a8917134ff2` 转走了 `max(uint256) - 1` 数量代币。 攻击者之所以先向自己地址转入 `248daac` ，目的是为了绕过 `batchTransfers` 函数中 `require(canPay(msg.sender, totalAmount));` 的限制。

最新 3 ERC-20 代币 转移 事件

交易哈希值	块龄	发送方		接收方	价值	代币
0xa11ad2e8575e00...	20 小时 3 分钟前	0xf4c01cb969dbd5...	出	0x1a2228110522c1...	0.000000000000000002	Distributed ... (DAAC)
0xa11ad2e8575e00...	20 小时 3 分钟前	0xf4c01cb969dbd5...	出	0x5bdd5dae928270...	115,792,089,237,316,000,000,000,00...	Distributed ... (DAAC)
0x98b9e15041566a...	1 天 3 小时前	0x6485fc77be2186f...	进	0xf4c01cb969dbd5...	248	Distributed ... (DAAC)

[下载 CSV导出]



分析总结

攻击者首先向自己的地址 `0xf4c01Cb969DBD5cA21DBD82016f289612d904731` 转入 `248daac`，绕过 `batchTransfers` 函数中 `require(canPay(msg.sender, totalAmount));` 的限制，然后利用 `batchTransfers` 函数中无符号算术溢出，向地址 `0x5bdd5dae9282700c88f3437df60a8a8917134ff2` 转走了 `max(uint256) - 1` 数量代币。

思考

DAAC智能合约中，本身已经写了 `SafeMath` 函数用于安全算术运算，但在 `batchTransfers` 函数中却并未使用，而是直接使用了 `+` 运算符，造成了这次代币攻击。说明开发者的安全意识还是不够。只在 `transfer` 函数中使用了 `SafeMath` 。

参考链接

[1] DAAC

https://etherscan.io/address/0xA4112bE97aca5b0cAbf5e1EFB35C99A0459B30c2

[2] Go

无符号整数溢出回绕问题

极光@知道创宇