

## 一、简介

Super Sushi Samurai 是一款在电报上的小游戏，只需要游玩该游戏即可获取项目方空投。3月22日，Super Sushi Samurai 声称受到了黑客攻击，导致 1310 个 ETH 不翼而飞，损失达到了 460 万美元。攻击是由于智能合约代码中的一个安全漏洞，允许未经授权的攻击者启动无限铸币功能，导致大量代币的创建，随后被出售到流动性池中，导致代币价值下跌了 99%。3月24日，该代币价值已归零。

## 二、游戏介绍

就是一个电报上的点击小游戏，可以看下这个视频：



Super Sushi Samurai.mp4

## 三、漏洞分析

### 1、查看智能合约

区块链上的交易、转账等支付功能基本都是通过智能合约实现的。与我们平时测试的项目不同的一点在于，区块链的智能合约代码全是公开的，因此通过智能合约审计，经验强的黑客很容易发现智能合约中存在的漏洞。我们也可以通过区块链浏览器去查看这个项目的智能合约：

合约地址：0xdfDCdbC789b56F99B0d0692d14DBC61906D9Deed

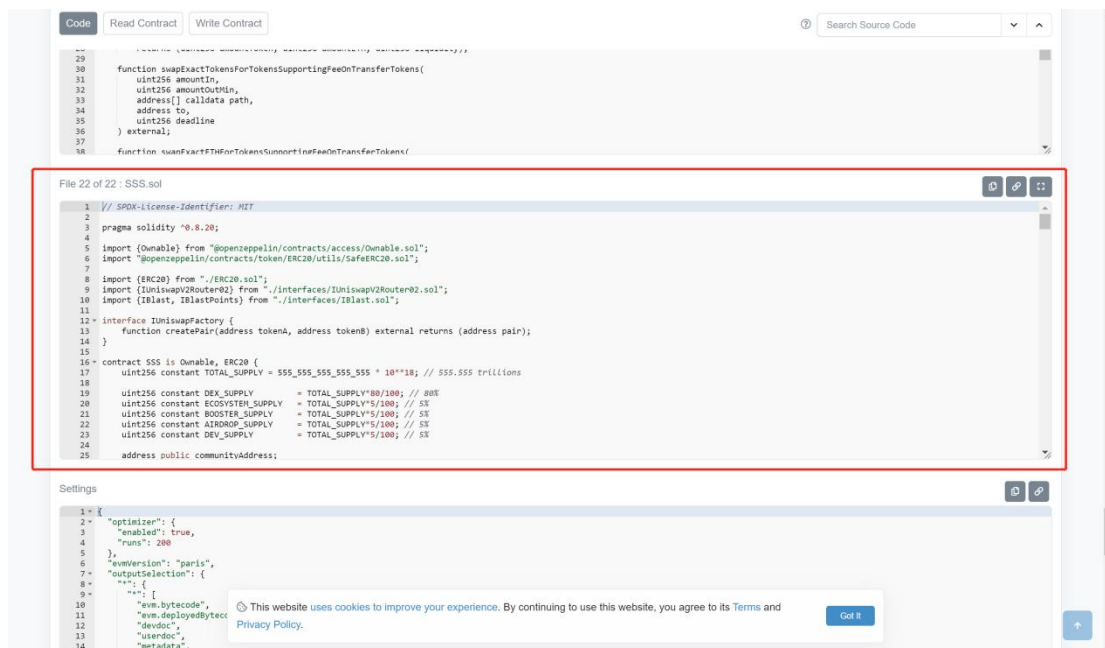
区块链浏览器：<https://blastscan.io/>

The screenshot shows the Blastscan.io interface. At the top, there's a search bar and navigation links. The main header displays the contract address: 0xdfDCdbC789b56F99B0d0692d14DBC61906D9Deed. Below this, there's a 'Contract Overview' section with fields for Balance (0.001201 ETH), Ether Value (\$4.18), and Token (\$0.00). To the right, there's a 'More Info' section with fields for My Name Tag, Contract Creator, and Token Tracker. Below these sections, there's a navigation bar with tabs for Transactions, Internal Txns, ERC20 Token Txns, Contract, Events, and Comments. The 'Contract' tab is selected, and a table of transactions is displayed below it. The table has columns for Txn Hash, Method, Block, Age, From, To, Value, and Txn Fee. The first transaction is an 'Approve' transaction from 0xb9a966be5c18ea220c... to 0xdfdcdbC789b56f99b0d... with a value of 0 ETH and a fee of 0.000034043543.

点击 Contract 就可以查看智能合约代码。

### 2、智能合约代码分析

存在漏洞的文件是 SSS.sol。sol 文件是 solidity 脚本语言写的，solidity 有点像前端的 js 代码，是智能合约中最常用的一种的语言。

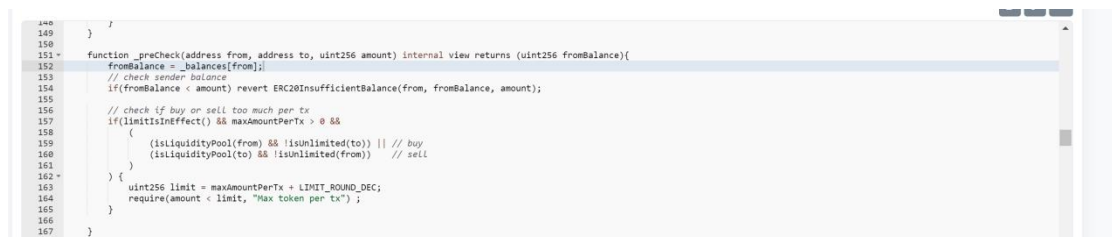


定位到 111 行的 `_update` 函数，该函数实现了转账功能，就是从传入的 `from` 地址转账到 `to` 地址，数量为 `amount` 代币。

113-116 行，检查 `from`、`to`、`amount` 是否为 0，如果是则调用父合约的 `update` 函数。这里检查的是否是铸币或者销毁代币，如果是，就不进行 `SSS.sol` 的转账功能。

118 行 `_botCheck()` 是通过监测交易时间来检查是否是机器人操作，这里我们不关注这个。

119 行的 `_preCheck()` 函数，我们跟进一下。



获取发送方余额，存在 `fromBalance` 参数里并返回该参数。

153 行是检查发送的代币要小于发送方的余额，否则会触发异常。

157-164 行是检查是否存在交易限制，检查每笔交易金额是否超过交易最大值，如果超过了会触发异常。

最终 `_preCheck()` 函数会返回发送方的余额并赋值给 `fromBalanceBeforeTransfer`



121 行是收取 `gas` 费用的操作，在区块链的交易中，每次发生交易都会收取 `gas` 费，这个费用一般是用来奖励支持这些区块的矿工，因为他们提供了算力。

我们跟进一下 122 行的 `_postCheck` 函数。

```

169 function _postCheck(address from, address to, uint256 amount) internal view returns (uint256 toBalance){
170     // check if buyer have too much token
171     toBalance = _balances[to] + amount;
172     if(limitIsInEffect() && maxAmountPerAccount > 0 &&
173         ((isLiquidityPool(from) && !isUnlimited(to))) // buy
174     ) {
175         uint256 limit = maxAmountPerAccount + LIMIT_ROUND_DEC;
176         require(toBalance < limit, "Max token per account");
177     }
178 }

```

获取接收方的余额，并加上转账的代币数量，存在 `toBalance` 参数里并返回该参数。

171 行就是计算交易执行后，接收方的余额。但是这里没有实际交易，只是先计算。

172-177 行也是检查交易最大限制，如果超过了会触发异常

最终 `_postCheck` 函数会返回接收方的余额并赋值给 `toBalance`，这个漏洞也存在于这个地方。

File 22 of 22 : SSS.sol

```

110 }
111 function _update(address from, address to, uint256 amount) internal override virtual {
112     // don't check if it is minting or burning
113     if (from == address(0) || to == address(0) || to == address(0xdead)) {
114         super._update(from, to, amount);
115         return;
116     }
117     _botCheck(from, to);
118     uint256 fromBalanceBeforeTransfer = _preCheck(from, to, amount);
119     uint256 amountAfterTax = amount - taxApply(from, to, amount);
120     uint256 toBalance = _postCheck(from, to, amountAfterTax);
121     _balances[from] = fromBalanceBeforeTransfer - amount;
122     _balances[to] = toBalance;
123     _unlockTokenForDev(from, to, amount);
124     emit Transfer(from, to, amountAfterTax);
125 }
126 // Buy too fast after init pool is bot
127 function _botCheck(address from, address to) internal {
128 }

```

124 行是计算发送方转账后余额并更新余额。

125 行是更新接收方余额。

127 行调用 `_unlockTokenForDev()` 函数来为开发者解锁代币，这个主要是为了控制代币的释放数量，防止一次性释放大额代币对市场造成不利影响，这里我们不去考虑这个。

129 行触发 `Transfer` 事件，通知链上的监听器有一笔转账发生。

### 3、双重消费漏洞分析

漏洞的成因在 122-125 行：

```

110 }
111 function _update(address from, address to, uint256 amount) internal override virtual {
112     // don't check if it is minting or burning
113     if (from == address(0) || to == address(0) || to == address(0xdead)) {
114         super._update(from, to, amount);
115         return;
116     }
117     _botCheck(from, to);
118     uint256 fromBalanceBeforeTransfer = _preCheck(from, to, amount);
119     uint256 amountAfterTax = amount - taxApply(from, to, amount);
120     uint256 toBalance = _postCheck(from, to, amountAfterTax);
121     _balances[from] = fromBalanceBeforeTransfer - amount;
122     _balances[to] = toBalance;
123     _unlockTokenForDev(from, to, amount);
124     emit Transfer(from, to, amountAfterTax);
125 }
126 }

```

这里乍一看没什么问题，我们以一个例子举例：

假如 `from` 地址和 `to` 地址为同一个地址，余额为 1000，并且转账所有余额。

```

111 function _update(address from, address to, uint256 amount) internal override virtual {
112     // don't check if it is minting or burning
113     if (from == address(0) || to == address(0) || to == address(0xdead)) {
114         super._update(from, to, amount);
115         return;
116     }
117
118     _botCheck(from, to);
119     uint256 fromBalanceBeforeTransfer = _preCheck(from, to, amount); 获取发送方余额:1000
120
121     uint256 amountAfterTax = amount - _taxApply(from, to, amount);
122     uint256 toBalance = _postCheck(from, to, amountAfterTax); 获取接收方交易后余额: 2000
123
124     _balances[from] = fromBalanceBeforeTransfer - amount; from余额变成0, to余额变成2000
125     _balances[to] = toBalance;
126
127     _unlockTokenForDev(from, to, amount);
128
129     emit Transfer(from, to, amountAfterTax);
130 }
131

```


如果 from 和 to 为同一个地址，那么交易后，该地址的余额就从 1000 翻倍成 2000 了。


（这里我们不考虑 gas 费消耗的问题，因为一般都很低，大概千分之几）

漏洞的利用手法就是自己给自己转账，余额就可以不断翻倍。

#### 四、英雄出现

这个漏洞万幸是被一名白帽子发现的，他利用该漏洞拯救了 1310 个 ETH，并返回给项目方。不幸的是仍然有 40 个 ETH 被黑客出售。



**Super Sushi Samurai | SSS**


@SSS\_HQ

1. Post-mortem:

The token contract has a bug where transferring your entire balance to yourself doubles it. h/t [@coffeecoin](#)

2. Damage details:

total eth in pool before exploit: 1339.50 ETH

Whitehat: 1,310.04 ETH

Blackhat : 40.28 ETH

we remove LP and got: 29.09 ETH

3. Update:

White hat has been co-operative, we are working out a plan that doesn't affect the white hat and the users. We hope negotiations can conclude soon. We would like to thank [@samczsun](#) and Seal 911 for helping with this process.

由 Google 翻译自 英语

1. 尸检:

代币合约有一个错误，将您的全部余额转移给自己会使余额翻倍。小时/吨 [@coffeecoin](#)

2. 损坏详情:

漏洞利用前池中的 ETH 总量: 1339.50 ETH



白帽: 1,310.04 ETH

黑帽: 40.28 ETH

我们删除 LP 并得到: 29.09 ETH

3.更新:

白帽一直在配合，我们正在制定一个不影响白帽和用户的计划。我们希望谈判能够尽快结束。我们要感谢[@samczsun](#)和 Seal 911 在这一过程中提供的帮助。

翻译得准确吗？请提供反馈，以便我们加以改进：  

下午2:00 · 2024年3月22日 · 9.7万 查看