BlockSec Follow
Jan 2, 2021 · 3 min read · Listen

Deposit Less, Get More: yCREDIT Attack Details

By BlockSec Team, Zhejiang University, China

On January 2, 2021 (Beijing Time 07:25am), our monitoring system ThunderForecast reported a series of suspicious transactions towards the yCREDIT smart contract. Then, we used the EthScope system developed by our research team to analyze these transactions and confirmed that all reported transactions are malicious. Note that, the vulnerability was also disclosed in Twitter (Link1, Link2). In this blog, we illustrate the attack details.

Details

The attack is due to the number of tokens minted is inconsistent with the intended one. As such, the attacker can get more numbers of yCREDIT tokens with lower price. Then these tokens can be sold to gain profits.

The vulnerable function is in the `_deposit` function of the StableYieldCredit contract.

In the following, we will use an attack transaction to illustrate the whole process.

```
479     function _deposit(IERC20 token, uint amount) internal {
480         uint _value = LINK.getPriceUSD(address(token)) * amount / uint256(10)**token.decimals();
481         require(_value > 0, "!value");
482 
483         (address _pair, uint amountA,) = _addLiquidity(address(token), address(this), amount, _value)
            ;
484 
485         token.safeTransferFrom(msg.sender, _pair, amountA);
486         _mint(_pair, _value); // Amount of scUSD to mint
487 
488         uint _liquidity = ISushiswapV2Pair(_pair).mint(address(this));
489         collateral[msg.sender][address(token)] += _liquidity;
490 
491         collateralCredit[msg.sender][address(token)] += _value;
492         uint _fee = _value * FEE / BASE;
493         _mint(msg.sender, _value - _fee);
494         _mint(address(this), _fee);
495         notifyFeeAmount(_fee);
496 
497         emit Deposit(msg.sender, address(token), _value, amount, _value);
498     }
```

The attacker first transferred *1e-8 WBTC* and *331.335 yCredit* tokens to the *WBTC-yCREDIT Pair pool*. Then the attacker deposited *0.5 WBTC* to the *StableYieldCredit* contract to launch the attack.

Specifically, the `_value` is calculated using the `amount` (0.5) of the `token` (0x2260fac5e5542a773aa44fbcfedf7c193bc2c599 - WBTC) based on the price oracle provider ChainLink (line 480, _value is `1466786010075`). The intention is to calculate the value of the deposited WBTC in USD. Then the contract will transfer the number of yCREDIT tokens (`_value - fee`) to the one who deposited the WBTC (the attacker). That's because the value of yCREDIT equals one USD (as designed by the system). Everything is fine except the attacker loses a small amount of `fee`.

Moreover, the contract will add the deposited WBTC to the *WBTC-yCREDIT Pair pool*. That's because if the deposited WBTC is locked into the contract, it will lose liquidity. As such, it first calculated the value of the token pair (WBTC to yCREDIT) that will be put into the pool. This value is calculated using the function `_addLiquidity`. Basically, it is calculated based on the existing reserves inside the pool. Since the pool only has *1e-8 WBTC* and *331.335 yCREDIT* tokens, the `amountA` calculated is `44` (amountB is `1466786010075`). That means the attacker only spends `44e-8` WBTC (line 485) and gets `14667.86010075 - fee = 14594.52080025` yCREDIT tokens (line `493`). At the same time, there is a small number of WBTC (`1e-8 + 44e-8`) and (`331.335 + 14667.86010075`) yCREDIT tokens leaving in the pool.

To get profits, the attacker can simply trade the gained `14594.52080025` yCREDIT tokens in exchanges. Interestingly, the process to gain profits in this transaction is far more complicated than necessary. We have also observed a clever attacking strategy in other transactions.

There are a serial of transactions involved in the attack, including (but not limited to) the following ones.

Transaction	Profit
0x552168ceac3be761642a284e305d7738a67419d91f10c30aa2b9b12bb3b9fcaa	0.5412482639
0x9fdd41efbaf06a887140df3d981530f9d396ed31a0231953e308f420f2734f6d	0.8604772677
0xbd677cf54675ec699c27fa3d1d12639d59e94393ae02717c03ea86a0b6bfb3e1	2.303752271
0xfa6f468d5b183465c6ef982d9ae1f97a52c957e481bd6f06f0b4b42ef32d0ef6	2.852401265
0xf29748b3f83bb22664134be8dd6d4703a4f6e80f07bf3d34ce404b0411cdbff6	1.606121816
0x795ca3c12090d03c4d839bfc9d1c023bd5dfec7e34a3f865d5e9e54e781a60f7	3.343862102
0x9078024591df4329c8a56f328fa05ee0fc0ee7a11c54884e3a8cb12e01184ca6	4.028426744
0xdf3deac40a049218a46c83deb7eeca6733de4bfbb406807a8c937270c8d8ed6d	5.006161716
0x78579a1f276ae224c127982d43cf312259f032efed4f51b7f9a1f8ff277269f1	7.104836196
0xa4ebcb5f26cacb6422528edc7d19e0a167a4f9c30a7b1e8066dbf21b66344811	10.35846317
0xa14a22f5e232e4c3318fc39f92777ca449d64ac851391feab2d33970418dbd49	6.449939611
0x50a24650557728d39354ab57e206a1703499f89b841559257ae1cc16a917b79f	13.05652598

Update

2020/01/03: There is an new smart contract that fixes the vulnerability.

Timeline

- 2021/01/02 07:25 Beijing Time, attacks were captured by our system
- 2021/01/03 00:36 Beijing Time, this blog is released

Smart Contract Defi Attack Security