Analysis of the Euler Finance's 197M Exploit — the Largest Security Incident so far in Q1 2023

On Mar 13, 2023, DeFi protocol Euler Finance was exploited with a ~$197M loss in multiple Txs. Around 34,224,863 $USDC, 849 $WBTC, 85,818 $stETH and 8,877,507 $DAI were stolen.

About Euler Finance

Back in June 2020, Euler won first place in the Spark College Hackathon online competition organized by Encode Club.

Euler is initially a lending protocol built on top of Aave, Compound and other lending protocols that allows users to create their own lending marketplace for any ERC-20 Token, while also providing Reactive rate models to reduce governance intervention.

In August 2021, Euler received $8 million in Series A funding led by Paradigm. On the official website, Euler also shows that they have six auditors, and none of them have found critical vulnerabilities.

Related Info

Project address:

0x27182842e098f60e3d576794a5bffb0777e025d3

Attack Txs:

https://etherscan.io/tx/0xc310a0affe2169d1f6feec1c63dbc7f7c62a887fa48795d327d4d2da2d6b111d

https://etherscan.io/tx/0x47ac3527d02e6b9631c77fad1cdee7bfa77a8a7bfd4880dccbda5146ace4088f

https://etherscan.io/tx/0x71a908be0bef6174bccc3d493becdfd28395d78898e355d451cb52f7bac38617

https://etherscan.io/tx/0x62bd3d31a7b75c098ccf28bc4d4af8c4a191b4b9e451fab4232258079e8b18c4

https://etherscan.io/tx/0x465a6780145f1efe3ab52f94c006065575712d2003d83d85481f3d110ed131d9

https://etherscan.io/tx/0x3097830e9921e4063d334acb82f6a79374f76f0b1a8f857e89b89bc58df1f311

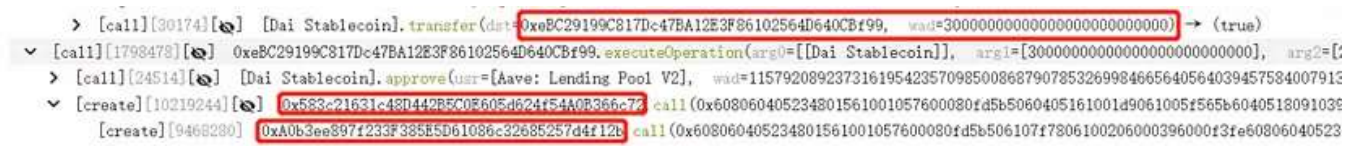
Attacker:

0x5f259d0b76665c337c6104145894f4d1d2758b8c

0xb2698c2d99ad2c302a95a8db26b08d17a77cedd4

We take one tx to analyze.

1. The hacker first flashloaned $30M $DAI and created two attack contracts. The 0x583 is for borrowing and the 0xA03 for liquidation. Then he deposited $300M to the borrowing contract.



2. The attacker deposited $20M to get 19,568,124 eDAI.

3. Then the attacker called mint function to use the 19,568,124 eDAIs to borrow 195,681,243 eDAIs (collateral assets) and 200,000,000 dDAIs (debt assets), thus scaling up the eDAI balance to ten times.

4. The attacker then deposited the remaining 10M DAI via the repay function, borrowing 195,681,243 eDAI and 200,000,000 dDAI again.

5. The attacker then carried out a donateToReserves operation to burn 100M eDAI, making eDAI < dDAI, which reaches the liquidation condition.

6. The liquidation contract liquidated the borrowing contract.

7. The attacker finally withdrew all 38.9M DAI from the contract and returned 30M DAI to AAVE, making a profit of ~8.9M $DAI.

Vulnerability Analysis

The Etoken contract's donateToReserves function fails to check the actual number of tokens held by the user and the health status of the user's ledger after donation, resulting in the attacker being able to donate 100 million eDAI (obtained through leverage, with the user actually depositing only 30 million DAI). After the donation, the health status of the user's ledger qualifies for liquidation, leading to the lending contract being liquidated. The lending contract transfers the eDAI and dDAI to the liquidation contract, which is then liquidated. Due to the unusually large amount of bad debt, the liquidation

contract will be liquidated with the maximum discount, resulting in 310.93M eDAI and 259.31M dDAI after the liquidation. At this point, the user has been restored to health and can withdraw funds, and the amount that can be withdrawn is the difference between eDAI and dDAI. However, since there are only 38.9M DAI in the pool, only that amount can be withdrawn.

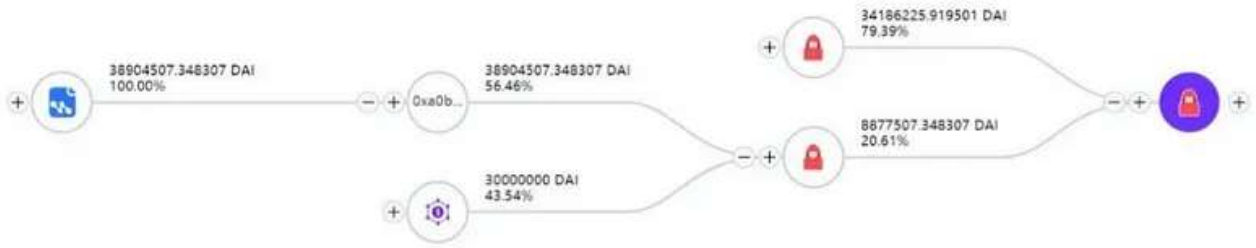
Overall, the root cause of the attack is that the Etoken contract does not properly check the actual number of tokens held by the user and the health status of the user's ledger after donation, which provides an opportunity for attackers.

```
354    function donateToReserves(uint subAccountId, uint amount) external nonReentrant {
355        (address underlying, AssetStorage storage assetStorage, address proxyAddr, address msgSender) = CALLER();
356        address account = getSubAccount(msgSender, subAccountId);
357
358        updateAverageLiquidity(account);
359        emit RequestDonate(account, amount);
360
361        AssetCache memory assetCache = loadAssetCache(underlying, assetStorage);
362
363        uint origBalance = assetStorage.users[account].balance;
364        uint newBalance;
365
366        if (amount == type(uint).max) {
367            amount = origBalance;
368            newBalance = 0;
369        } else {
370            require(origBalance >= amount, "e/insufficient-balance");
371            unchecked { newBalance = origBalance - amount; }
372        }
373
374        assetStorage.users[account].balance = encodeAmount(newBalance);
375        assetStorage.reserveBalance = assetCache.reserveBalance = encodeSmallAmount(assetCache.reserveBalance + amount);
376
377        emit Withdraw(assetCache.underlying, account, amount);
378        emitViaProxy_Transfer(proxyAddr, account, address(0), amount);
379
380        logAssetStatus(assetCache);
381    }
382 }
```

Fund Flow

As of press time, 80,080.98 ETH are held at address 0xb2698c2d99ad2c302a95a8db26b08d17a77cedd4.

88,651.70 ETH and 43063733.27 DAI are held at address 0xb66cd966670d962C227B3EABA30a872DbFb995db.



Only a small number (100 ETH) were transferred to Tornado Cash.

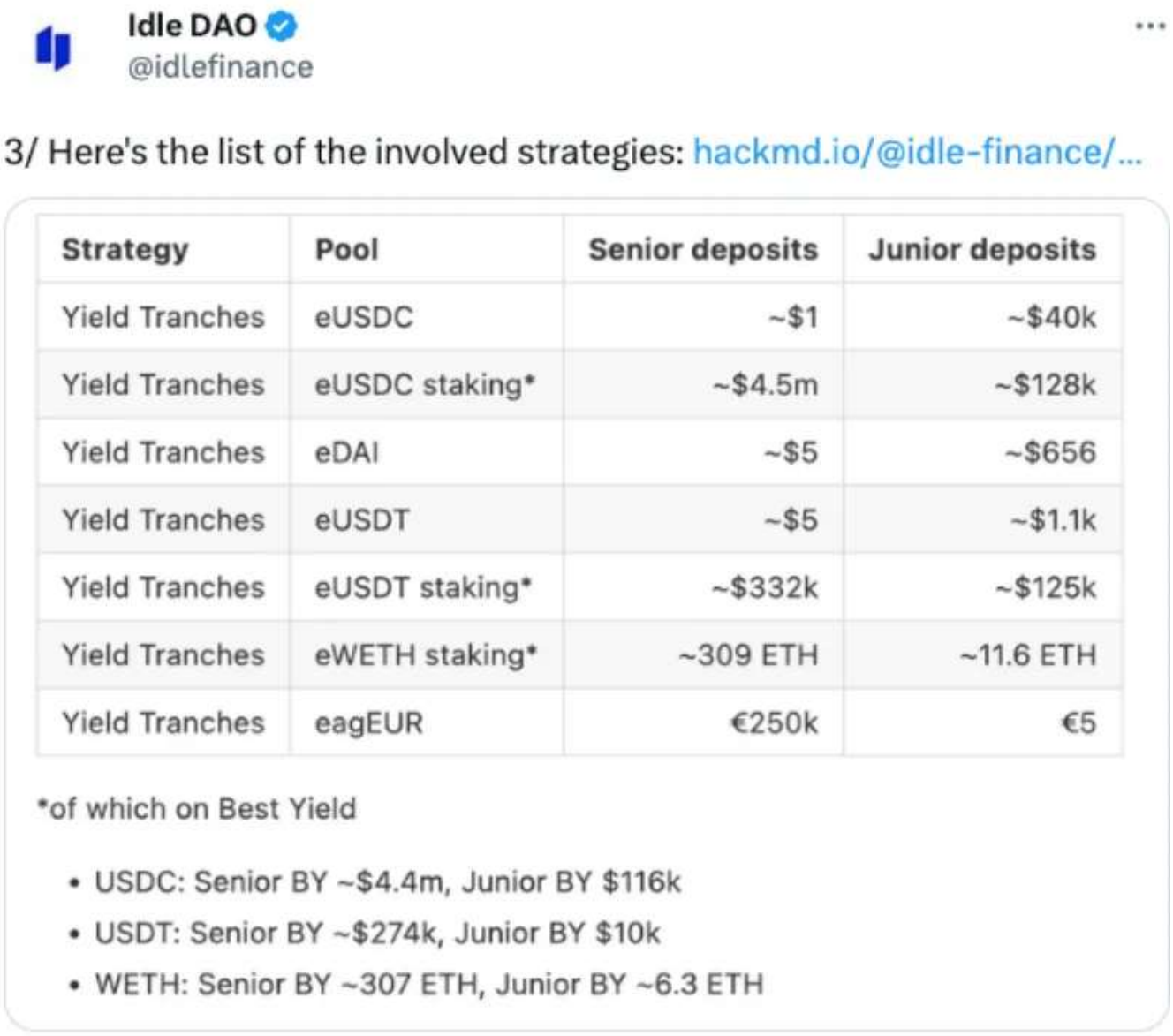
Try Beosin KYT at kyt.beosin.com.

What are the protocols affected by Euler?

According to 0xScope, Angle Protocol has $17.6 million USDC in Euler; Idle DAO has $4.6 million USDC in Euler; SwissBorg has 6,357 Ether and 1.7 million USDT deposited in Euler. After the attack, SwissBorg quickly borrowed 4,752 cbETH to mitigate losses, but still had about $4.26 million assets in Euler; the 0x28a5 whale address and czsamsun.eth had $4 million and $2.74 million in the protocol, respectively.

Yield aggregator Yearn tweeted that while it was not directly exposed to the Euler attack, some Yearn vaults were indirectly exposed to the hack.

Idle Finance tweeted that it was also affected.



Recommendations

1. In developing smart contracts, note whether the subsequent addition of functionality has an impact on the preceding logic.

2. Be careful to check on the assets.