Are Your Funds Safe? Multiple Projects Attacked Due to Vyper Reentrancy Vulnerability with a total loss of more than $59 Million

Beosin · Follow

5 min read · Jul 31

Listen Share

On the evening of July 30th, 2023, multiple projects encountered a dark moment.

At around 21:35 on July 30th, according to Beosin's EagleEye security risk monitoring, the NFT lending protocol JPEG'd was attacked.

While the Beosin security team was analyzing the situation, several other projects were attacked in succession.

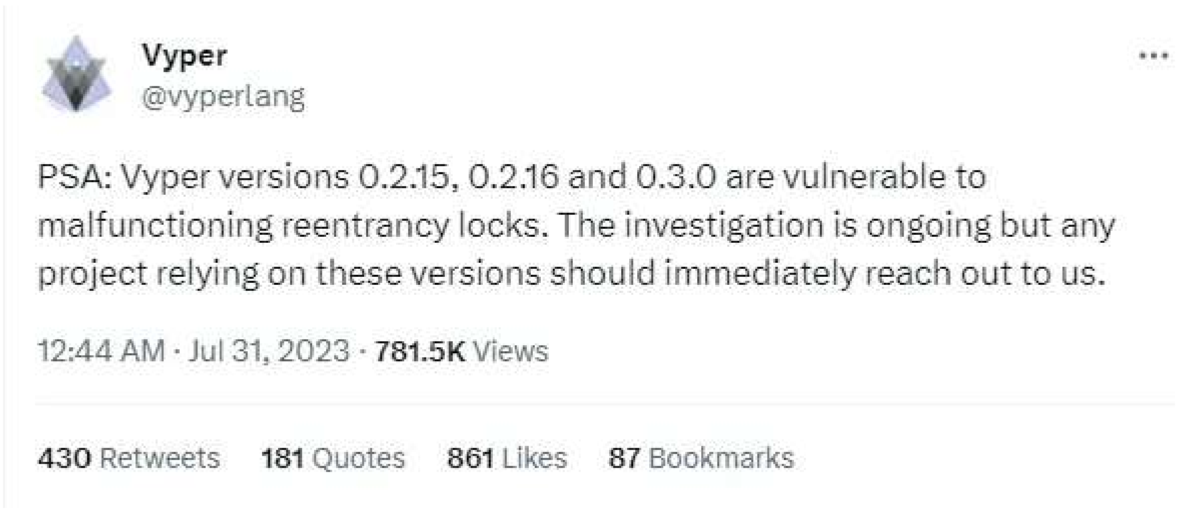
At around 22:51 on July 30th, the msETH-ETH pool was raided by hackers.

At around 23:35 on July 30th, the alETH-ETH pool was also cracked using the same attack method.

Shortly after, liquidity pools belonging to DeFi projects Alchemix and Metronome were successively attacked.

The root reason of these multiple attacks is Vyper?

According to a tweet from the Ethereum programming language Vyper on July 31st, **Vyper versions 0.2.15, 0.2.16, and 0.3.0 have vulnerabilities in their reentrancy locks. Combined with the ability of native ETH to call callback function during transfers, liquidity pools with native ETH created by these versions can be hacked by reentrancy attacks.**



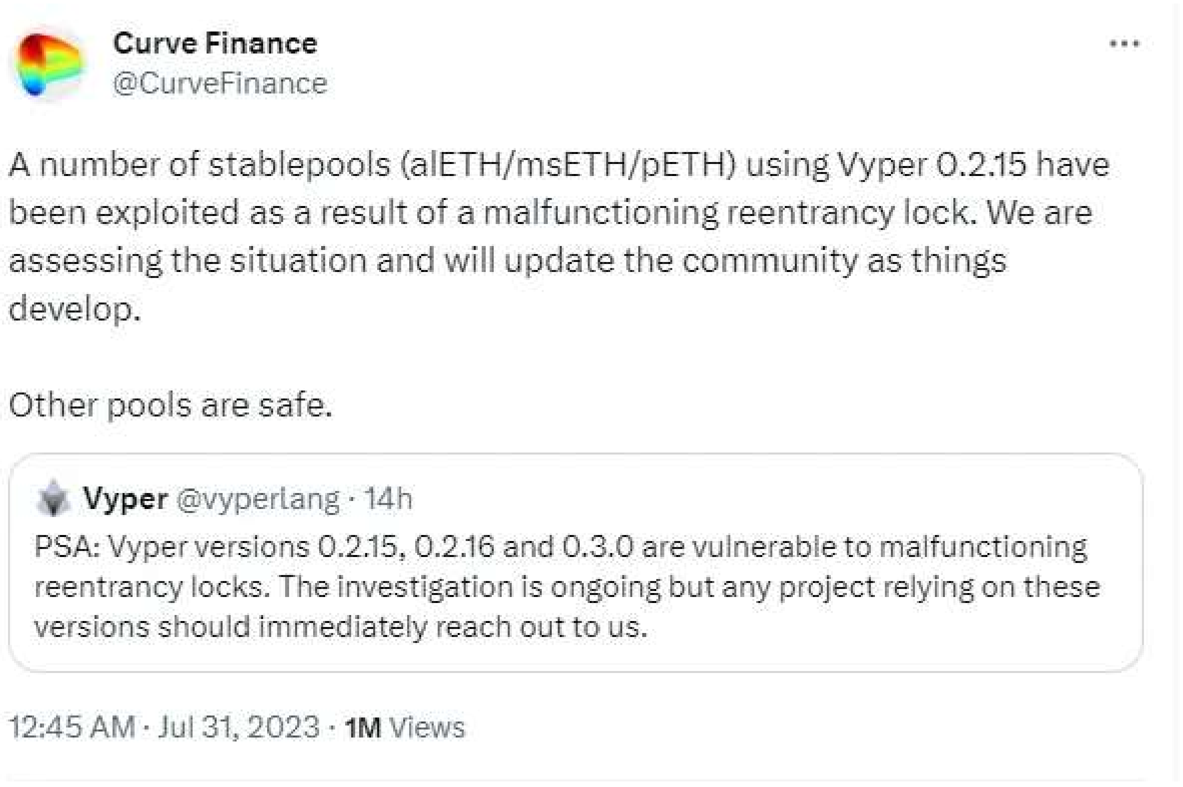
Vyper
@vyperlang

PSA: Vyper versions 0.2.15, 0.2.16 and 0.3.0 are vulnerable to malfunctioning reentrancy locks. The investigation is ongoing but any project relying on these versions should immediately reach out to us.

12:44 AM · Jul 31, 2023 · **781.5K** Views

430 Retweets **181** Quotes **861** Likes **87** Bookmarks

Curve's official Twitter then stated that many stablecoin pools (alETH/msETH/pETH) using Vyper 0.2.15 were attacked due to reentrancy lock failures, but other pools are safe.


Curve Finance
@CurveFinance

A number of stablepools (alETH/msETH/pETH) using Vyper 0.2.15 have been exploited as a result of a malfunctioning reentrancy lock. We are assessing the situation and will update the community as things develop.

Other pools are safe.

Vyper @vyperlang · 14h
PSA: Vyper versions 0.2.15, 0.2.16 and 0.3.0 are vulnerable to malfunctioning reentrancy locks. The investigation is ongoing but any project relying on these versions should immediately reach out to us.

12:45 AM · Jul 31, 2023 · 1M Views

Analysis by Beosin Security Team on the Attacked Projects.

Here are the relevant transactions related to the hacking incident:

●Attack Transaction:

0xc93eb238ff42632525e990119d3edc7775299a70b56e54d83ec4f53736400964
0xb676d789bb8b66a08105c844a49c2bcffb400e5c1cfabd4bc30cca4bff3c9801

0xa84aa065ce61dbb1eb50ab6ae67fc31a9da50dd2c74eefd561661bfce2f1620c

0x2e7dc8b2fb7e25fd00ed9565dcc0ad4546363171d5e00f196d48103983ae477c

0xcd99fadd7e28a42a063e07d9d86f67c88e10a7afe5921bd28cd1124924ae2052

●Attacker's Address

0xC0ffeEBABE5D496B2DDE509f9fa189C25cF29671

0xdce5d6b41c32f578f875efffc0d422c57a75d7d8

0x6Ec21d1868743a44318c3C259a6d4953F9978538

0xb752DeF3a1fDEd45d6c4b9F4A8F18E645b41b324

●Attacked Contracts

0xc897b98272AA23714464Ea2A0Bd5180f1B8C0025

0xC4C319E2D4d66CcA4464C0c2B32c9Bd23ebe784e

0x9848482da3Ee3076165ce6497eDA906E66bB85C5

0x8301AE4fc9c624d1D396cbDAa1ed877821D7C511

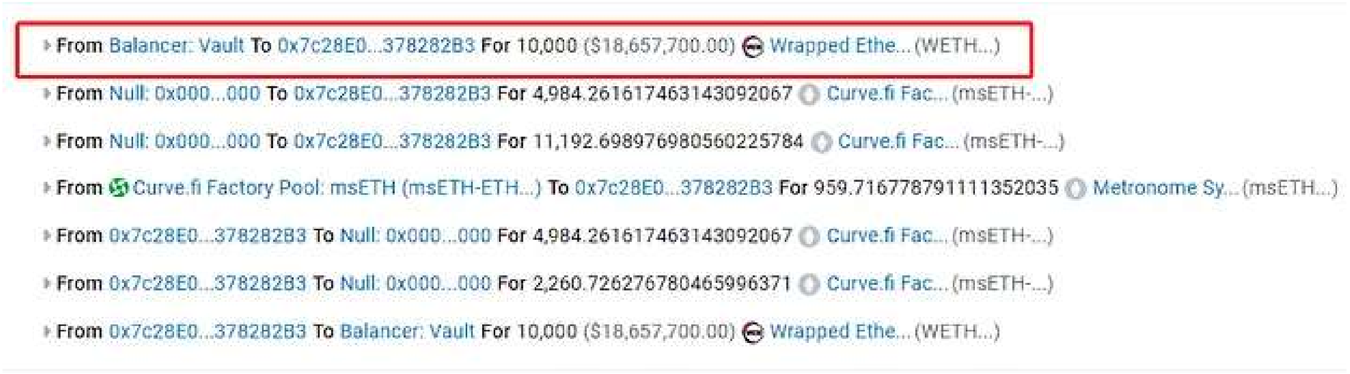
Vulnerability Analysis

According to Beosin security team, the main cause of this attack was the failure of the reentrancy lock in Vyper 0.2.15. Attackers added liquidity by reentering the add_liquidity function when removing liquidity using the remove_liquidity function of the related liquidity pools. Due to the balance update occurring before reentry into the add_liquidity function, price calculation errors occurred.

Attack Process

We take the msETH-ETH-f pool attacked by transaction 0xc93eb238f as an example.

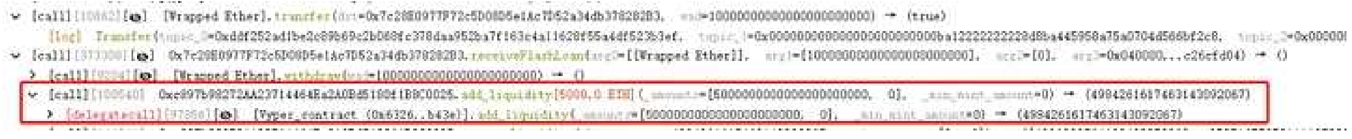
In the preparation stage, the hacker first flash loaned 10,000 ETH through the Balancer:Vault as attack funds.



```
From Balancer: Vault To 0x7c28E0...378282B3 For 10,000 ($18,657,700.00) Wrapped Ethe... (WETH...)
From Null: 0x000...000 To 0x7c28E0...378282B3 For 4,984.261617463143092067 Curve.fi Fac... (msETH-...)
From Null: 0x000...000 To 0x7c28E0...378282B3 For 11,192.698976980560225784 Curve.fi Fac... (msETH-...)
From Curve.fi Factory Pool: msETH (msETH-ETH...) To 0x7c28E0...378282B3 For 959.716778791111352035 Metronome Sy... (msETH...)
From 0x7c28E0...378282B3 To Null: 0x000...000 For 4,984.261617463143092067 Curve.fi Fac... (msETH-...)
From 0x7c28E0...378282B3 To Null: 0x000...000 For 2,260.726276780465996371 Curve.fi Fac... (msETH-...)
From 0x7c28E0...378282B3 To Balancer: Vault For 10,000 ($18,657,700.00) Wrapped Ethe... (WETH...)
```

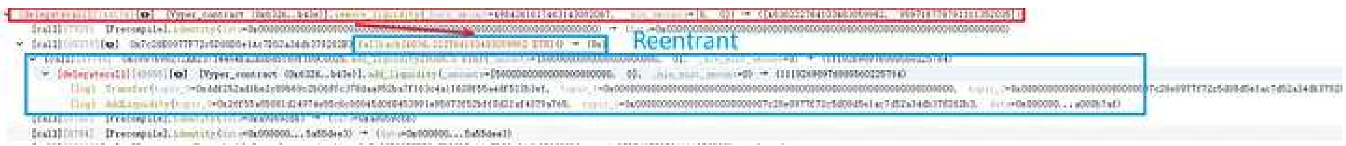
Attack Stage

1. In the first step, the attacker called the add_liquidity function to add the 5000 ETH flash loan to the pool.



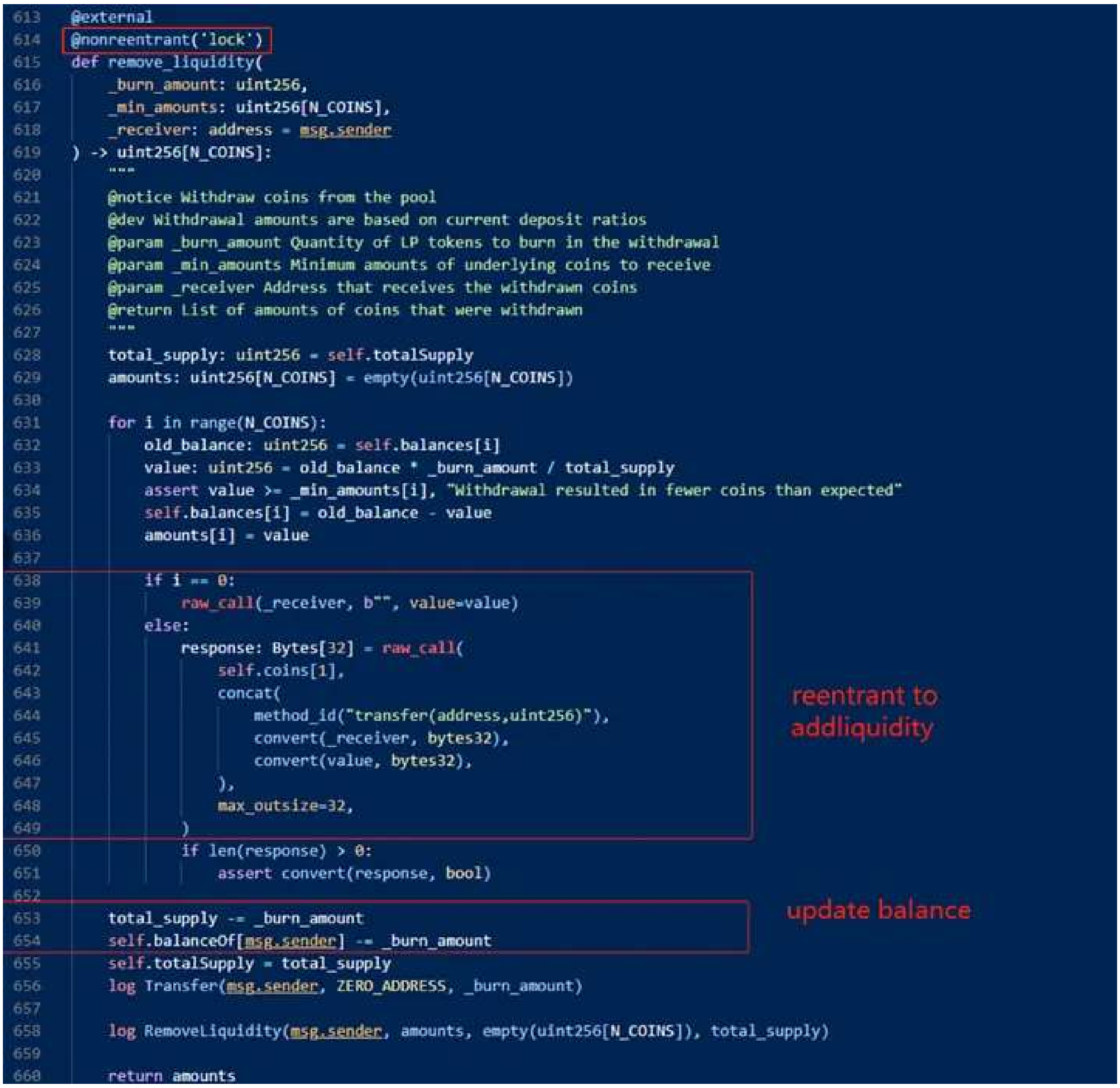
```
[call][10862] [Wrapped Ether].transfer(dst=0x7c28E0977F72c5D08D5e1Ac7D52a34db378282B3, wad=10000000000000000000000) → (true)
    [log] Transfer(topic_0=0xddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f55a4df523b3ef, topic_1=0x000000000000000000000000ba12222222228d8ba445958a75a0704d566bf2c8, topic_2=0x00000
[call][373300] 0x7c28E0977F72c5D08D5e1Ac7D52a34db378282B3.receiveFlashLoan(arg0=[[Wrapped Ether]], arg1=[10000000000000000000000], arg2=[0], arg3=0x040000...c26cfd04) → ()
  [call][9394] [Wrapped Ether].withdraw(wad=10000000000000000000000) → ()
  [call][100540] 0xc897b98272AA23714464Ea2A0Bd5180f1B8C0025.add_liquidity[5000,0 ETH](_amounts=[5000000000000000000000, 0], _min_mint_amount=0) → (4984261617463143092067)
    [delegatecall][97360] [Vyper_contract (0x6326..b43e)].add_liquidity(_amounts=[5000000000000000000000, 0], _min_mint_amount=0) → (4984261617463143092067)
```

2. In the second step, the attacker called the remove_liquidity function to remove the ETH liquidity from the pool and then reentered the add_liquidity function to add liquidity.

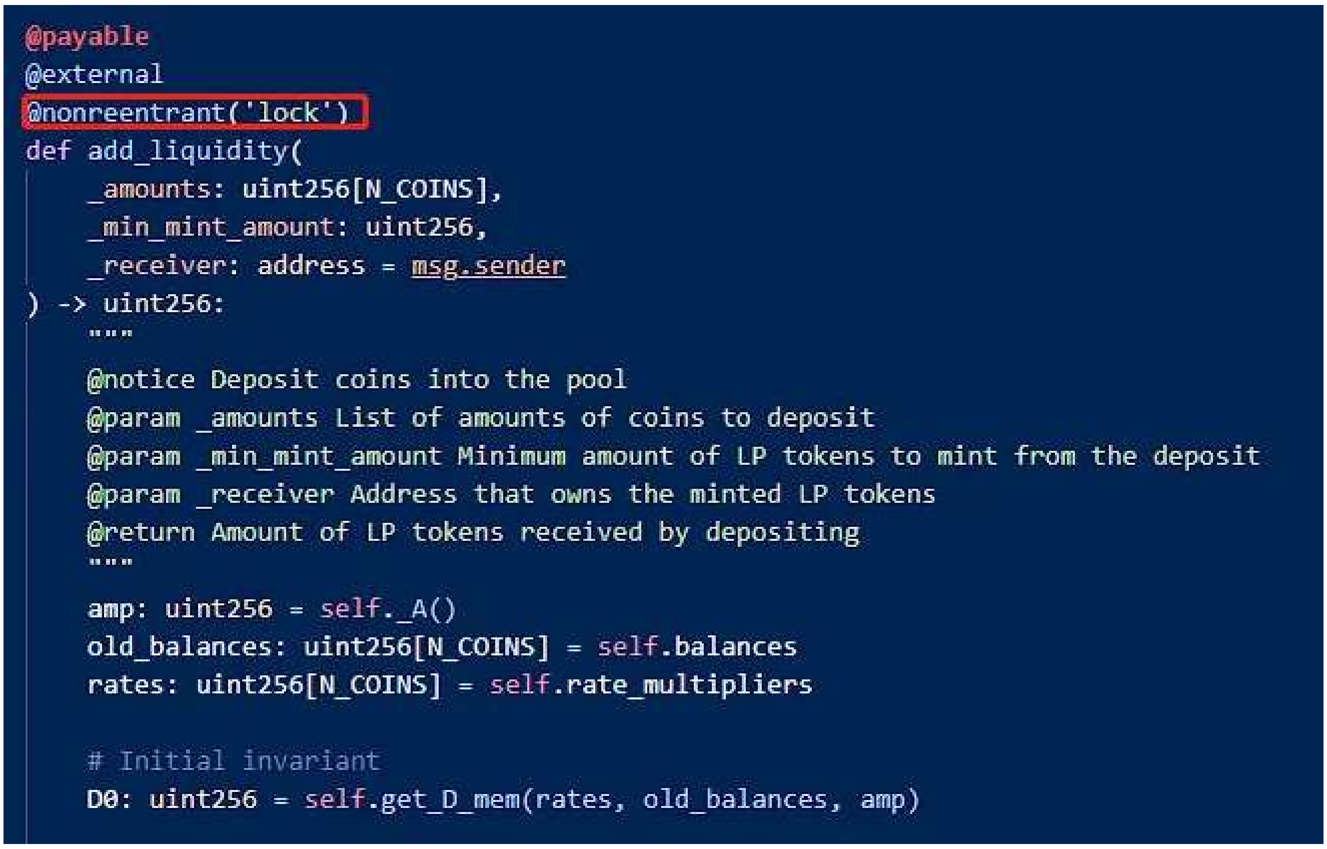


[illegible]

3. Due to the balance update occurring before reentry into the add_liquidity function, price calculation errors occurred. It is worth noting that both the remove_liquidity function and the add_liquidity function have used reentrancy locks to prevent reentry.

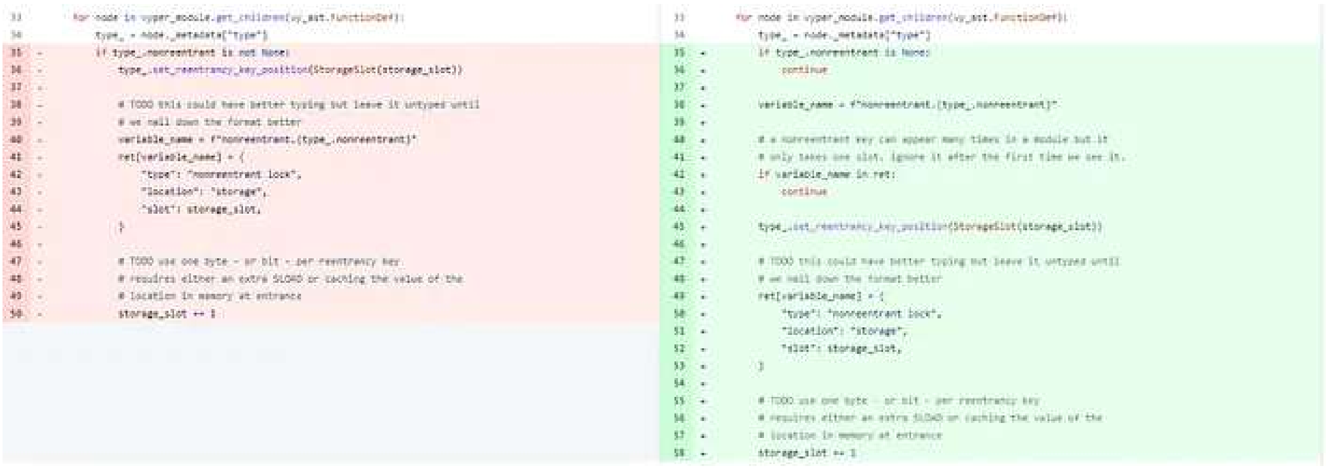


```
@external
@nonreentrant('lock')
def remove_liquidity(
    _burn_amount: uint256,
    _min_amounts: uint256[N_COINS],
    _receiver: address = msg.sender
) -> uint256[N_COINS]:
    """
    @notice Withdraw coins from the pool
    @dev Withdrawal amounts are based on current deposit ratios
    @param _burn_amount Quantity of LP tokens to burn in the withdrawal
    @param _min_amounts Minimum amounts of underlying coins to receive
    @param _receiver Address that receives the withdrawn coins
    @return List of amounts of coins that were withdrawn
    """
    total_supply: uint256 = self.totalSupply
    amounts: uint256[N_COINS] = empty(uint256[N_COINS])

    for i in range(N_COINS):
        old_balance: uint256 = self.balances[i]
        value: uint256 = old_balance * _burn_amount / total_supply
        assert value >= _min_amounts[i], "Withdrawal resulted in fewer coins than expected"
        self.balances[i] = old_balance - value
        amounts[i] = value

        if i == 0:
            raw_call(_receiver, b"", value=value)
        else:
            response: Bytes[32] = raw_call(
                self.coins[1],
                concat(
                    method_id("transfer(address,uint256)"),
                    convert(_receiver, bytes32),
                    convert(value, bytes32),
                ),
                max_outsize=32,
            )
            if len(response) > 0:
                assert convert(response, bool)

    total_supply -= _burn_amount
    self.balanceOf[msg.sender] -= _burn_amount
    self.totalSupply = total_supply
    log Transfer(msg.sender, ZERO_ADDRESS, _burn_amount)

    log RemoveLiquidity(msg.sender, amounts, empty(uint256[N_COINS]), total_supply)

    return amounts
```



```
@payable
@external
@nonreentrant('lock')
def add_liquidity(
    _amounts: uint256[N_COINS],
    _min_mint_amount: uint256,
    _receiver: address = msg.sender
) -> uint256:
    """
    @notice Deposit coins into the pool
    @param _amounts List of amounts of coins to deposit
    @param _min_mint_amount Minimum amount of LP tokens to mint from the deposit
    @param _receiver Address that owns the minted LP tokens
    @return Amount of LP tokens received by depositing
    """
    amp: uint256 = self._A()
    old_balances: uint256[N_COINS] = self.balances
    rates: uint256[N_COINS] = self.rate_multipliers

    # Initial invariant
    D0: uint256 = self.get_D_mem(rates, old_balances, amp)
```

4. Therefore, the reentrancy lock was not effective here. By reading the vulnerable Vyper code shown below, it can be found that when the name of the reentrancy lock appears for the second time, the original number of storage_slot will increase by 1. In other words, the slot that originally acquired the lock is 0, but after another function uses the lock, the slot becomes 1, and the reentrancy lock already fails.



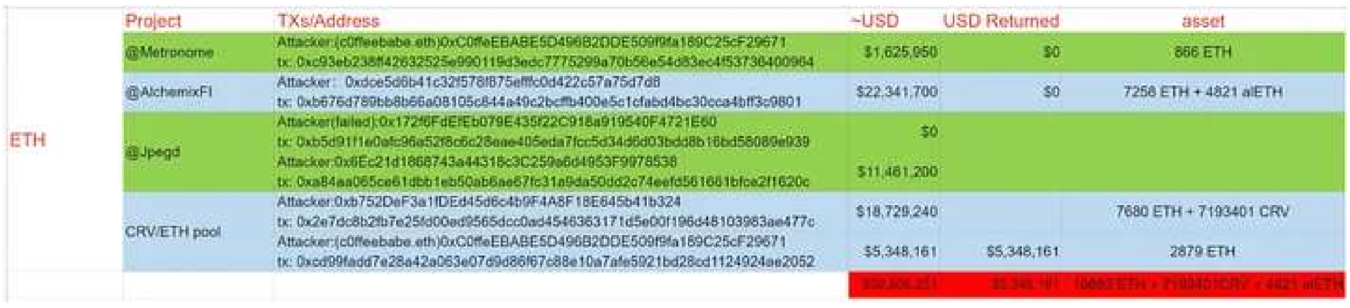
```
    for node in vyper_module.get_children(vy_ast.FunctionDef):
        type_ = node._metadata["type"]
-       if type_.nonreentrant is not None:
-           type_.set_reentrancy_key_position(StorageSlot(storage_slot))
-
-           # TODO this could have better typing but leave it untyped until
-           # we nail down the format better
-           variable_name = f"nonreentrant.{type_.nonreentrant}"
-           ret[variable_name] = {
-               "type": "nonreentrant lock",
-               "location": "storage",
-               "slot": storage_slot,
-           }
-
-           # TODO use one byte - or bit - per reentrancy key
-           # requires either an extra SLOAD or caching the value of the
-           # location in memory at entrance
-           storage_slot += 1
```

```
    for node in vyper_module.get_children(vy_ast.FunctionDef):
        type_ = node._metadata["type"]
+       if type_.nonreentrant is None:
+           continue
+
+       variable_name = f"nonreentrant.{type_.nonreentrant}"
+
+       # a nonreentrant key can appear many times in a module but it
+       # only takes one slot. ignore it after the first time we see it.
+       if variable_name in ret:
+           continue
+
+       type_.set_reentrancy_key_position(StorageSlot(storage_slot))
+
+       # TODO this could have better typing but leave it untyped until
+       # we nail down the format better
+       ret[variable_name] = {
+           "type": "nonreentrant lock",
+           "location": "storage",
+           "slot": storage_slot,
+       }
+
+       # TODO use one byte - or bit - per reentrancy key
+       # requires either an extra SLOAD or caching the value of the
+       # location in memory at entrance
+       storage_slot += 1
```

https://github.com/vyperlang/vyper/commit/eae0eaf86eb462746e4867352126f6c1dd43302f

Funds Statistics

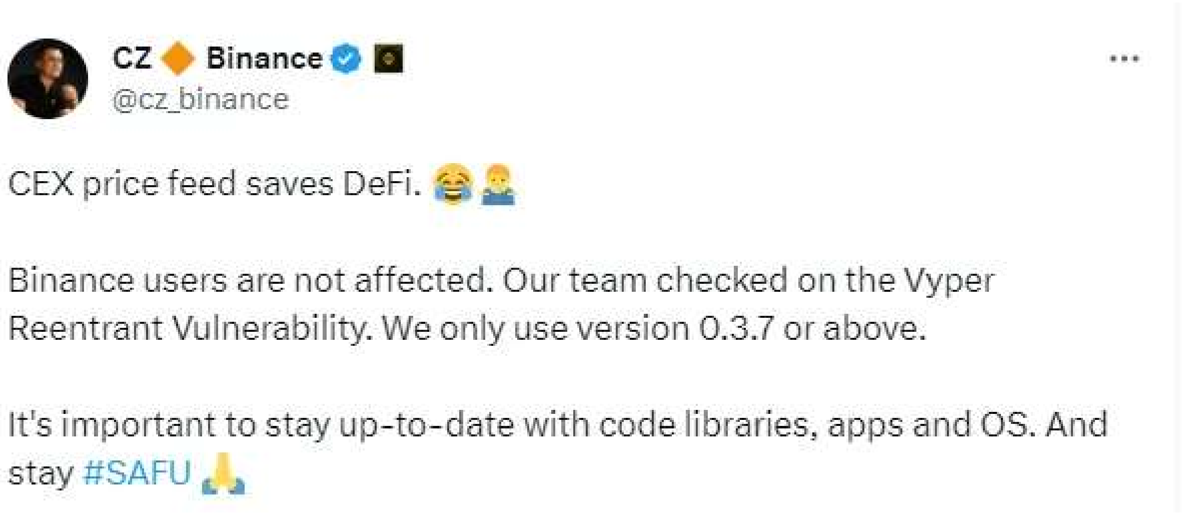
At the time of publication, **the funds lost in this attack have exceeded $59 million. Beosin KYT has monitored that the c0ffeebabe.eth address has returned 2879 ETH, but the stolen funds remain on multiple attacker addresses.**



	Project	TXs/Address	~USD	USD Returned	asset
ETH	@Metronome	Attacker:(c0ffeebabe.eth)0xC0ffeEBABE5D496B2DDE509f9fa189C25cF29671 tx: 0xc93eb238ff42632525e990119d3edc7775299a70b56e54d83ec4f53736400964	$1,625,950	$0	866 ETH
	@AlchemixFI	Attacker: 0xdce5d6b41c32f578f875efffc0d422c57a75d7d8 tx: 0xb676d789bb8b66a08105c844a49c2bcffb400e5c1cfabd4bc30cca4bff3c9801	$22,341,700	$0	7258 ETH + 4821 alETH
	@Jpegd	Attacker(failed):0x172f6FdEfEb079E435f22C918a919540F4721E60 tx: 0xb5d91f1e0afc96a52f8c6c28eae405eda7fcc5d34d6d03bdd8b16bd58089e939	$0		
		Attacker:0x6Ec21d1868743a44318c3C259a6d4953F9978538 tx: 0xa84aa065ce61dbb1eb50ab6ae67fc31a9da50dd2c74eefd561661bfce2f1620c	$11,461,200		
	CRV/ETH pool	Attacker:0xb752DeF3a1fDEd45d6c4b9F4A8F18E645b41b324 tx: 0x2e7dc8b2fb7e25fd00ed9565dcc0ad4546363171d5e00f196d48103983ae477c	$18,729,240		7680 ETH + 7193401 CRV
		Attacker:(c0ffeebabe.eth)0xC0ffeEBABE5D496B2DDE509f9fa189C25cF29671 tx: 0xcd99fadd7e28a42a063e07d9d86f67c88e10a7afe5921bd28cd1124924ae2052	$5,348,161	$5,348,161	2879 ETH
			$59,506,251	$5,348,161	[illegible] ETH + 7193401CRV + 4821 alETH

Subsequent Impact

Regarding the impact of this event, on July 31st, Binance founder CZ tweeted that "CEX price feed saves DeFi." Binance users were not affected, and the Binance team checked on the Vyper reentrancy vulnerability. Binance only uses version 0.3.7 or above. It is important to keep the codebase, applications, and operating systems up to date.



CZ 🔶 Binance
@cz_binance

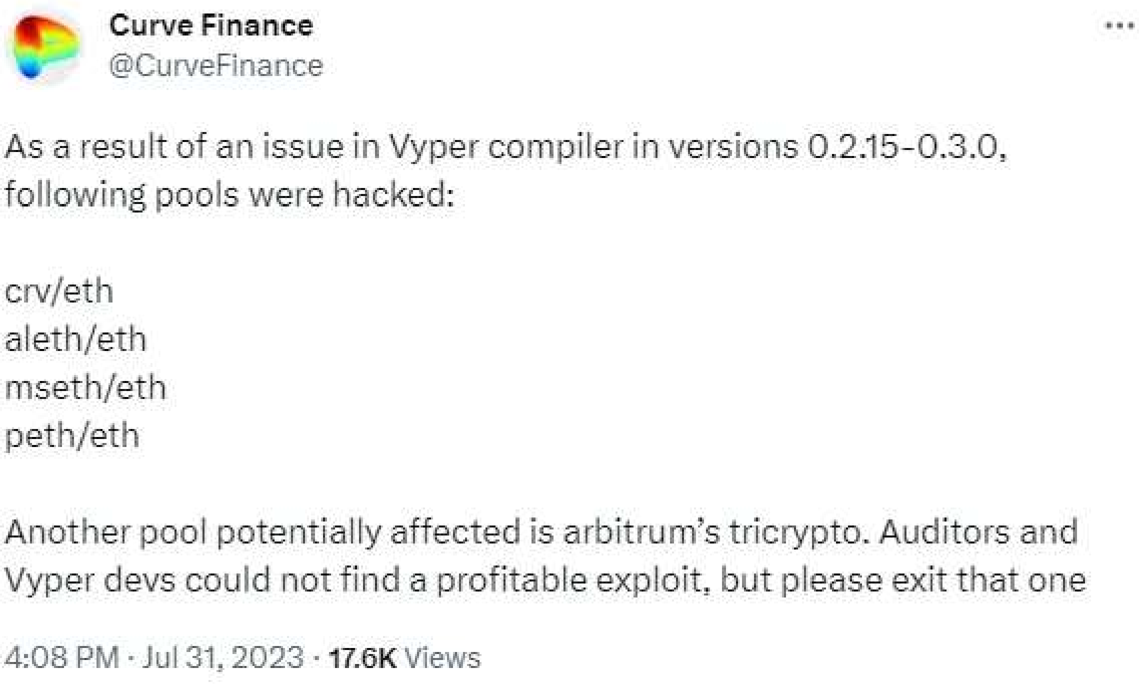
CEX price feed saves DeFi. 😂🤷‍♂️

Binance users are not affected. Our team checked on the Vyper Reentrant Vulnerability. We only use version 0.3.7 or above.

It's important to stay up-to-date with code libraries, apps and OS. And stay #SAFU 🙏

On July 31st, Curve tweeted that due to problems with the Vyper compiler in versions 0.2.15–0.3.0, CRV/ETH, alETH/ETH, msETH/ETH, and pETH/ETH were attacked by hackers. Additionally**, the Arbitrum Tricrypto pool may also be affected. Auditors and Vyper developers have not yet found exploitable vulnerabilities, but users are advised to remove their liquidity from the pool.**

It can be seen that the impact of this event has not yet ended, and users who have funds in these pools need to pay more attention.


Curve Finance
@CurveFinance

As a result of an issue in Vyper compiler in versions 0.2.15-0.3.0, following pools were hacked:

crv/eth
aleth/eth
mseth/eth
peth/eth

Another pool potentially affected is arbitrum's tricrypto. Auditors and Vyper devs could not find a profitable exploit, but please exit that one

4:08 PM · Jul 31, 2023 · 17.6K Views

Regarding this event, Beosin security team recommends that **the reentrancy locks in Vyper versions 0.2.15, 0.2.16, and 0.3.0 all fail and related projects are advised to check for themselves. After a project is launched, it is strongly recommended that the project team continues to pay attention to vulnerability disclosures of third-party components/dependency libraries and timely avoid security risks.**

Beosin is a leading global blockchain security company co-founded by several professors from world-renowned universities and there are 40+ PhDs in the team, and set up offices in 10+ cities including Hong Kong, Singapore, Tokyo and Miami. With the mission of "Securing Blockchain Ecosystem", Beosin provides "All-in-one" blockchain security solution covering Smart Contract Audit, Risk Monitoring & Alert, KYT/AML, and Crypto Tracing. Beosin has already provided security for 2000+ blockchain companies, audited more than 3000 smart contracts and protected our customers' assets worth of $500 billion.