Anatomy of a Bridge Reserve Smart Contract Vulnerability (and how we fixed it)

Kyber Network
Feb 8 · 8 min read

Smart contact bugs and vulnerabilities are an inevitable part of the crypto space. Building applications on a novel new platform like the blockchain introduces many new complexities, and with billions of dollars worth of value stored on Ethereum, learning and sharing these vulnerabilities through security audits are vital to advancing this space.

In this blog post we'll introduce you to a Kyber reserve manager smart contract vulnerability discovered by samczsun a few months ago, that opened up the three Kyber-run DEX bridge reserves to a potential attack vector on the Oasis bridge reserve, and how we fixed it (thankfully without the exploit being used). The blog does get technical as we'd like Ethereum developers to take as much from it as possible.

The Vulnerability

On the 13th of September, samczsun, an independent security researcher, reached out to us to give us details on his discovery of a potential attack vector on the Oasis bridge reserve. The attack vector was a sophisticated exploit that places re-entrancy function calls in the middle of Kyber's trade process, thus allowing an attacker to temporarily

manipulate the best conversion rate and then ask the reserve to send the tokens according to that conversion rate from its internal inventory.

After a thorough investigation, we discovered that using the same exploitation model, this could potentially also have been used to attack the Uniswap bridge reserve and PT (Promo Token) reserve. In the worst case, the attack could have potentially drained the funds held on only those three Kyber-run reserves, and so our operations team immediately *disabled the feature to use internal funds on the Uniswap and Oasis reserves*, and withdrew most of the funds we had on the PT reserve.

To the best of our knowledge, **no** damage has been done and **no** users' funds or integrated platforms have been affected.

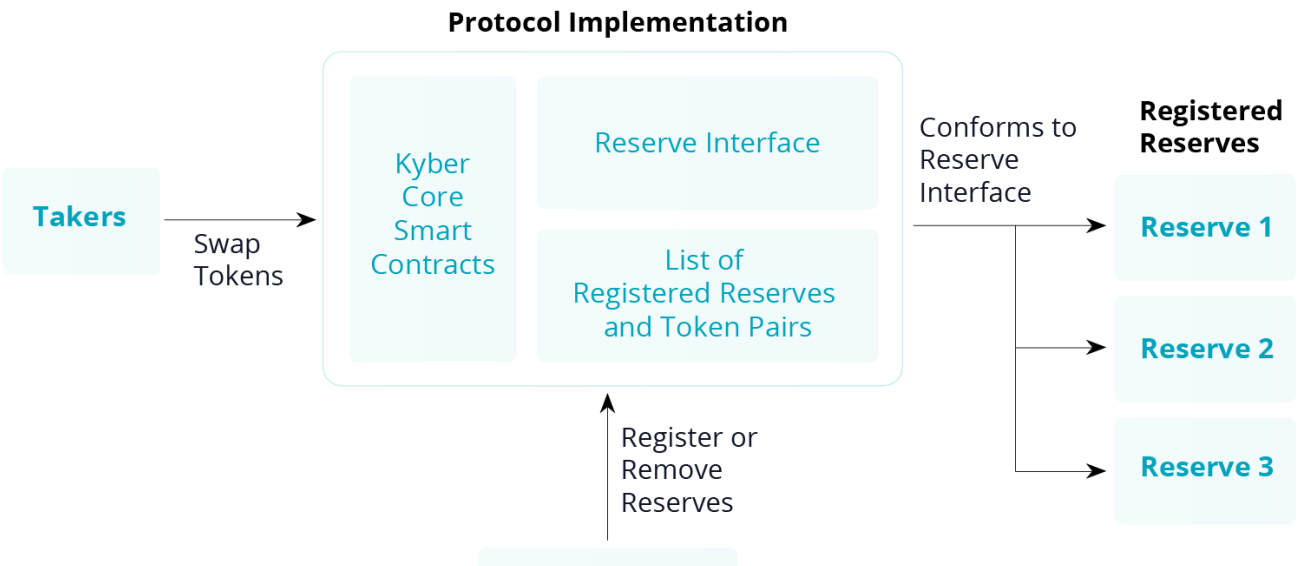
Before we get into the technicalities, it is important to stress that:

a) Users funds are safe and were never at risk
b) The only reserves affected are the three Kyber-operated reserves: *Uniswap-bridge reserve, Oasis-bridge reserve, and PT reserve (promo token reserve). No other reserves were impacted.*
c) The KNC token contract is not impacted in any way

To recognize his efforts in discovering the attack, and to show our appreciation for his responsible disclosure, we've provided samczsun with a bounty.

What is the scope of the vulnerability?

To understand the scope of the vulnerability, let us briefly refresh our memory on Kyber's high level smart contract design:



Maintainers

Network Overview

Multiple parties, such as takers, the protocol contract, reserves, and tokens interact with each other to provide the trade functionality. Takers and reserves hold funds, and the protocol contract makes the market and settles the trade by moving funds between takers and reserves.

How does the attack work?

The basic idea of the attack is to list a malicious token that is developed and controlled by the attacker on Kyber's permissionless orderbook reserve, and to be able to inject custom smart contract codes in the `decimals`, `transfer` and `transferFrom` functions. Those functions can be called in the middle of a trade, allowing the custom codes to be invoked to change some trade states when it is expected to be consistent throughout the settlement. The exact attack steps are described as follows:

Attacking the Oasis-bridge reserve

1. Create a custom ERC20 token, named tokenA, with an abnormal implementation of the `decimals`, `transfer` or `transferFrom` function. Let's say `transferFrom` is customized.

2. List tokenA on Kyber's permissionless orderbook reserve and put an order to buy 1 tokenA with 0.1 ETH.

3. The attacker calls `trade()` on Kyber to trade 1 tokenA to DAI (the more DAI Kyber returns the more profit the attacker gets).

4. Kyber will call `tokenA.transferFrom` to move the token to the network contract, in this first execution of `transferFrom`, it also creates an order on OasisDEX to buy 0.1 ETH with 1000 DAI. This order is not taken immediately because OasisDEX has that as a feature.

5. The network contract looks for the best rate for 1 tokenA to ETH and finds the permissionless order the attacker put, 1 tokenA = 0.1 ETH.

6. The network contract looks for the best rate for 0.1 ETH to DAI by iterating through the list of DAI reserves and get to the Oasis-bridge reserve.

7. The Oasis-bridge reserve gets the best order that can serve 0.1 ETH from OasisDEX for both sides (sell and buy) and sees there is an arbitrage because of the attacker's order of buying 0.1 ETH with 1000 DAI, it decides not to use its own funds. Instead, ***it thinks it will take the order from OasisDEX for the trade later*** so it returns the rate of 1000 DAI for 0.1 ETH.

8. Obviously, 1000 DAI is the best rate for 0.1 ETH so the network contract knows 1 tokenA can convert for 1000 DAI and proceeds with settlement.

9. First, it settles the trade of 1 tokenA for 0.1 ETH. The network contract asks the permissionless reserve to accept 1 tokenA and send back 0.1 ETH. The permissionless reserve calls `tokenA.transferFrom` to transfer 1 tokenA to itself. However, this `transferFrom` function knows this is the second time it is called and it is the owner of the malicious order it created in step 4, it cancels this order.

10. The network contract sends 0.1 ETH to the reserve and asks the Oasis-bridge reserve to send back 1000 DAI.

11. The Oasis-bridge reserve receives the command and proceeds to the trade, it gets the orderbook information from OasisDEX to decide if it should use its own internal funds or if it should take an Oasis order. Since the malicious order was canceled in the previous step, the orderbook is as per normal. Hence, the condition to use its own internal funds is met. It trusts the rate provided by itself in step 7 and just sends 1000 DAI back to the network contract.

12. The network contract sends 1000 DAI to the attacker. The attack is done.

Attacking the Uniswap-bridge reserve

The same idea to the Oasis-bridge reserve is applied here. However, instead of creating and canceling Oasis order in 1 transaction, it uses a bit of funds to temporarily skew the rate of a token on Uniswap, manipulating the rate of that token and Uniswap-bridge reserve uses that rate to perform the trade with its internal funds.

Attacking the PT reserve

The idea is similar to the Oasis-bridge reserve attack as it uses no-immediate-settle order feature of OasisDEX to temporarily manipulate the price of DAI/ETH thus manipulate the price of the PT token and after the trade (trade 1 PT for more ETH than normal) is done, it cancels the Oasis order without any costs.

Why the vulnerabilities work

The Oasis-bridge reserve did not double-check the rate sent by the network contract upon settlement. Hence, it was not able to spot that the OasisDEX orderbook has changed.

The Uniswap-bridge reserve relies on the Uniswap rate for its trades using its internal funds without knowing that it can be skewed and unskewed without any costs.

PT reserve relies on the OasisDEX order book to peg PT price to DAI price without knowing the OasisDEX order book can be temporarily changed without any costs.

Mitigations on each reserve

Oasis-bridge reserve mitigation

We realize that the reserve lacked a method to make the source of the token (internal funds or Oasis orderbook) to be consistent throughout step 7 and step 11. To mitigate this, we turn the leftmost bit of the returned rate in the step 7 to 1 to indicate that the source of the token to settle the trade must be the internal funds, otherwise, it must take from the Oasis orderbook.

With this mitigation, after the Oasis orderbook is canceled, the reserve will try to find the non-existing order and, critically, revert the whole trade.

Uniswap-bridge reserve mitigation

Having already turned off the internal fund feature so the attack does not impact any funds on the Uniswap-bridge reserve, we are already working on a new algorithm to bring back the internal funds feature without it being affected by the existing vulnerability.

PT reserve mitigation

Since the PT reserve is to serve PT tokens, a promotion token that Kyber issues to serve the team's marketing activities, and its fund is relatively small, we decided to trust Makerdao's DAI price feed and use it to peg PT price to DAI price.

With this mitigation, the attacker needs to manipulate Makerdao's DAI price feed to perform a similar attack.

Why this vulnerability was not discovered earlier

All 3 reserves use their own funds in combination with external DEXes to become an additional liquidity source, with the aim to generate profits, and use those profits later to provide better liquidity during more adverse market conditions.

We call them hybrid reserves as they do 2 jobs at the same time:

1. Source liquidity from other places to increase the whole network's liquidity
2. Provide more liquidity from their own funds when the market is more inefficient

These hybrid-models are in the experimental phase and therefore there are only limited funds in these reserves, so only internal audits were conducted. Going forward, to improve bug-detection coverage, we will conduct external audits on the reserves, on top of our own extensive internal audit.

What can we learn from this?

All of the 3 reserves use external smart contracts as a price oracle to determine usages of their own funds. It is important for the reserves to note that:

1. Price from an external oracle can be temporarily manipulated if there are external calls before the trade execution happens.
2. There are 2 steps in a trade. First a rate lookup, then a trade settlement. The reserve is required to return the rate in the first step and settle with that *exact rate* in the second step. As the settlement means the reserve has to transfer an amount of a token from somewhere to the network, it is important to be sure about the source of the token, the amount (or the rate) to send and the destination. Since the rate is passed by the network contract instead of the reserve itself, and there can be reentrancy in the middle of the 2 steps, double-checking on the rate and the source of the token is a recommended practice.

Does this vulnerability affect other DeFi projects integrated with Kyber?

No. However, part of the attack shows that it is possible and even easy to ***temporarily*** manipulate the price of a token on Oasis DEX and Uniswap DEX, hence also on Kyber Network.

There are a few DeFi projects that use Kyber as a price oracle, and they need to exercise caution. In our Developer portal (Price Feed Security), we recommend some best practices if one wants to use Kyber as a price feed. Please use it with care.

For any questions, please reach out to us on @kyberdeveloper

Ethereum Featured