CERTORA

Synthetix Bug in the Depot Contract

General Description

To allow the sUSD minters to easily sell their minted coins for ether, the Synthetix system provides a simple platform for depositing sUSD tokens into a deposit box ("Depot"), which will sell the tokens on behalf of them for ether in synthetix.exchange. The Depot implements a queue (deposits[]) to keep track of all the deposits (deposits[i].user & deposits[i].amount).

When someone wishes to exchange its ether for sUSD (exchangeEtherForSynths), the Depot iterates over the queue from its start: In the simple case, the first entry in the queue has more than enough sUSD to completely fulfill the requested amount. In that case, the Depot updates the record with the reduced amount (original_amount – requested_amount), transfers the ether of the sUSD to the entry's user, transfers the requested amount of synth to the sUSD buyer, and exits the function. In the more complex case, where there isn't enough sUSD in the first entry of the queue, the Depot pops the record out of the queue, transfers a proportional amount of ether to the entry's user (according to the deposit amount), transfers the synths in the entry to the sUSD buyer, reduces the amount of requested sUSD accordingly and continues to iterate to the next deposit entry. At the end, the requested sUSD will be equal 0, meaning that the request was fulfilled (the entire amount of ether was converted to sUSD).

However, it is possible that the amount of sUSD requested is larger than the sum of all deposits. In that case, the queue will be completely emptied, and the function's initiator will get back any amount of ether that was left (not converted to sUSD). Also, an sUSD depositor can decide she wishes to withdraw the amount she deposited. In that case, the Depot scans the entire queue, deleting deposit entries that belongs to the depositor (Nulling the user & amount fields of those entries). Therefore, it is possible that during the run of exchangeEtherForSynths, the Depot will encounter a null record. In that case, the Depot will simply skip that record.

The bug lies in the fact that during the ether transfer to the depositor (by a send function), a callback function can initiate another call to the Depot. With the queue correctly formed, it is possible to make the pointer to the start of the queue to skip over a valid deposit entry, thus making the owner of this entry to lose the money she deposited.

Technical Description and Example

When exchangeEtherForSynths is called, the Depot iterates over the queue, from depositStartIndex to depositEndIndex (both are global variable of the contract), or until the requested amount of synth is fulfilled. For every null entry or an entry without enough synths (which will be emptied soon), the Depot immediately advances depositStartIndex by 1.

Let's assume that Eve, Alice, Bob & Carol have all deposited 100 sUSD to the Depot (in that order). Afterwards, Alice changed her mind and withdrew her deposits. Therefore the deposit will look like this –
deposit[] = [{owner: Eve, amount:100}, {owner: "", amount:0}, {owner: Bob, amount:100}, {owner: Carol, amount:100}], with depositStartIndex=0, depositEndIndex=4.

Dave wishes to transfer his 150 sUSD worth of ether to sUSD, and he calls exchangeEtherForSynths. At first, since Eve's entry has only 100 sUSD, it will be deleted and the depositStartIndex will advanced by 1. During the ether send operation, Eve calls exchangeEtherForSynths once again (with 1 sUSD worth of ether). Now, since depositStartIndex=1, Depot will first encounter the empty entry left by Alice, so it will skip it, making depositStartIndex=2. The next entry (of Bob) has more than enough sUSD to fulfill Eve's request and the function will exit. Now, when we return to the original call (by Dave), the queue looks like this:
[{owner: "", amount:0}, {owner: "", amount:0}, {owner: Bob, amount:99}, {owner: Carol, amount:100}], with depositStartIndex=2. It should be noted that the next entry in the iteration will be yet again Alice's empty entry, so once again the Depot

advances depositStartIndex by 1, making depositStartIndex equal 3. In the next entry (by Bob), Dave request will be fulfilled (it only needed another 50 sUSD) and the function will exit. The final state of the Depot is problematic - The double counting of Alice's empty entry caused Depot to skip over Bob's entry (2) and now depositStartIndex points to Carol's entry (3).

In theory, this method allows us to skip an arbitrary amount of entries, according to the number of empty entries after the entry owned by Eve, even to the point where the depositStartIndex is larger than the depositEndIndex.

There is one major drawback – In order for the attack to work, we must be able to carry out the second call to exchangeEtherForSynths consuming only 21K gas (the amount of gas allowed for a callback). According to our tests, such call demands around 80k, which makes this attack unrealistic in the current gas metering scheme. However, it should be noted that it is possible that a different exploit might be able to utilize the bug under the current gas constraints.

Possible Mitigations

- Completely prevent callback execution in the contract.
- Change the for-loop into a while-loop and access the start of the queue each time with deposits[depositStartIndex], which is always up to date.
- Allow users to extract their synths from a specific index in the deposits array (even if it is outside of the queue range), this will help to ensure that even in a case where the consistency of the queue indices is violated, a depositor can still gain back the sUSD she deposited in a certain entry.