Taker Protocol Audit Report

Version 1

Gin

August 02, 2023

Taker Protocol Audit Report

Gin

August 02, 2023

Prepared by: Gin

Table of Contents

Protocol Summary

Decentralized Leveraged Trading on Wemix

Disclaimer

The security team makes all effort to find as many vulnerabilities in the code in the given time period, but holds no responsibilities for the findings provided in this document. A security audit by the team is not an endorsement of the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the Solidity implementation of the contracts.

Risk Classification

		Impact		
		High	Medium	Low
	High	H	H/M	M
Likelihood	Medium	H/M	M	M/L
	Low	M	M/L	L

We use the CodeHawks severity matrix to determine severity. See the documentation for more details.

Scope

All files in codebase

Issues found

Severity	Number of Issues Found
High	0
Medium	3

Severity	Number of Issues Found
Low	2
Info	0
Total	5

Findings

Medium-Risk Issues

[M-01] Use safeTransfer instead of transfer in TERC721 contract claimERC20Airdrop function

Description:

```
IERC20Upgradeable(vars.airdropTokenAddresses[typeIndex]).transfer(
    initiator, vars.airdropBalance)
IERC20Upgradeable(token).transfer(to, amount)
IERC20Upgradeable(vars.airdropTokenAddresses[typeIndex]).transfer(
    initiator, vars.airdropBalance)
```

In TERC721 contract claimERC20Airdrop function and AirdropFlashClaimReceiver contract executeOperation & transferERC20 function, transfer is used instead of safeTransfer The ERC20.transfer() functions return a boolean value indicating success. This parameter needs to be checked for success. Some tokens do not revert if the transfer failed but return false instead.

Recommendation: Use OpenZeppelin's SafeERC20 versions with the safeTransfer functions that handle the return value check as well as non-standard-compliant tokens

[M-02] Potential reorg attack in UserFlashclaimRegistry contract _createReceiver function

Description:

```
address payable receiver = payable(receiverImplemention.clone());
```

The clone function generates a random seed based on the current timestamp and uses this seed to create a new contract instance. However, since the timestamp can be predicted by attackers, they can modify the timestamp by executing a Reorg attack and recreate the same seed in the future, resulting in the creation of the same contract address. This means that attackers can deploy another contract at the same address before the contract creation, allowing them to execute malicious operations, such

as stealing funds, on the same address An attacker can steal funds via a reorg attack if a contract is funded within a few blocks of being created inside a factory

Recommendation: Utilize cloneDeterministic rather than clone To address this issue, Solidity introduced the cloneDeterministic function. This function allows specifying an explicit seed parameter during contract creation instead of using the timestamp as a random seed. By using an explicit seed, the contract's address and state become predictable and are not affected by the timestamp cloneDeterministic uses the opcode and a salt to deterministically deploy the clone. Using the same implementation and salt multiple times will revert since the clones cannot be deployed twice at the same address When using cloneDeterministic function, be careful of this issue below: 2023-04-caviar-findings

[M-03] PriceOracle will use the wrong price if the Chainlink registry returns price outside min/max range

Description:

```
1 (, int256 price, , , ) = _tokenAggregators[asset].latestRoundData();
```

Chainlink aggregators have a built in circuit breaker if the price of an asset goes outside of a predetermined price band. The result is that if an asset experiences a huge drop in value (i.e. LUNA crash) the price of the oracle will continue to return the minPrice instead of the actual price of the asset. This would allow user to continue borrowing with the asset but at the wrong price. This is exactly what happened to Venus on BSC when LUNA imploded. The wrong price may be returned in the event of a market crash, an adversary will then be able to borrow against the wrong price and incur bad debt to the protocol

Recommendation:

```
1 require(price >= minPrice && price <= maxPrice, "invalid price");
```

use the proper minPrice and maxPrice for each asset Also, please refer to this article for further use of chainlink interface, such as if contract will be deployed in layer 2, please check the sequencer is not down before asking for price https://0xmacro.com/blog/how-to-consume-chainlink-price-feeds-safely/

Low-Risk Issues

[L-01] price feed staleness in "TakerOracleGetter" contract "getReserveAssetPrice" function

Description:

```
require(address(_tokenAggregators[asset]) != address(0), "no price
    exists");
(, int256 price, , , ) = _tokenAggregators[asset].latestRoundData();
return uint256(price);
```

It does not check for price feed staleness. It has happened before - a feed stops updating the price and returns a stale one

Recommendation:

```
require (updatedAt >= block.timestamp - 3600, "stale price")
```

validate that no more than 1 hour(or any value you want to set) has passed from the updatedAt timestamp value returned from latestRoundData, otherwise the transaction will revert. Also, a backup oracle option is suggested

[L-02] Array length should be checked in AirdropFlashClaimReceiver contract executeOperation function

Description:

```
address[] calldata nftAssets,
uint256[][] calldata nftTokenIds,
```

Function executeOperation takes two dynamic arrays as param, nftAssets length should be equal to nftTokenIds length, but this function doesn't have input validation. Before calling this function, it is necessary to ensure that the lengths of these two arrays are equal so that NFT contract addresses can be correctly matched with their corresponding token IDs. If the lengths are not equal, it may lead to erroneous NFT operations and even result in contract execution failure.

Recommendation:

```
require(nftAssets.length == nftTokenIds.length, "nftAssets and
    nftTokenIds lengths do not match");
```

add input validation for two dynamic arrays