CANTINA

Inshallah Goldsand

Security Review

Cantina Managed review by:

Kaden, Security Researcher

Cccz, Security Researcher

October 10, 2024

Contents

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity	Description
Critical	*Must* fix as soon as possible (if already deployed).
High	Leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
Medium	Global losses <10% or losses to only a subset of users, but still unacceptable.
Low	Losses will be annoying but bearable. Applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.
Gas Optimization	Suggestions around gas saving practices.
Informational	Suggestions around best practices or readability.

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

Goldsand is a halal Lido alternative: it is Ethereum staking that removes interest-bearing transactions and maximizes yield.

From Aug 5th to Sep 28th the Cantina team conducted a review of goldsand-v1 on commit hash da1be927. The team identified a total of **16** issues in the following risk categories:

- Critical Risk: 0
- High Risk: 0
- Medium Risk: 1
- Low Risk: 5
- Gas Optimizations: 5
- Informational: 5

3 Findings

3.1 Medium Risk

3.1.1 `OPERATOR_ROLE` can withdraw ETH when the protocol is paused

Severity: Medium Risk

Context: Goldsand.sol#L324

Description: When the protocol is paused, `OPERATOR_ROLE` and users should not be able to perform any actions.

However, since the `whenNotPaused` modifier is not added to the `operatorWithdrawETHForUser()` function, `OPERATOR_ROLE` can withdraw ETH from the protocol when the protocol is paused.

Recommendation: It is recommended to add the `whenNotPaused` modifier to the `operatorWithdrawETH-ForUser()` function.

Inshallah: Fixed in commit 96b61cc1.

Cantina Managed: Fixed.

3.2 Low Risk

3.2.1 `supportsInterface()` should not support `type(Initializable).interfaceId`

Severity: Low Risk

Context: Goldsand.sol#L428, WithdrawalVault.sol#L131

Description: `type(Initializable).interfaceId` is `0x00000000`, i.e. Initializable doesn't have any interface.

When it is supported in `supportsInterface()`, it means `supportsInterface()` will support any contract without any interface.

Recommendation: It is recommended to remove support for `type(Initializable).interfaceId`.

Inshallah: Fixed in commit 96b61cc1.

Cantina Managed: Fixed.

3.2.2 ETH withdrawn from `WithdrawalVault` to `Goldsand` is unexpectedly accounted for in `funder-ToBalance` mapping

Severity: Low Risk

Context: Goldsand.sol#L164-L174

Description: In `Goldsand.sol`, we have `fallback` and `receive` functions to handle any incoming ETH transfers by calling `fund`, applying the transferred amount to the `msg.sender`'s `funderToBalance`:

```
/**
 * @notice Fallback function
 * @dev Handles Ether sent directly to the contract.
 */
fallback() external payable whenNotPaused {
    fund();
}

/**
 * @notice Receive function
 * @dev Handles Ether sent directly to the contract.
 */
receive() external payable whenNotPaused {
    fund();
}
```

```
function fund() public payable whenNotPaused {
    if (msg.value < minEthDeposit) {
        revert TooSmallDeposit();
    }
    funderToBalance[msg.sender] += msg.value;
    emit Funded(msg.sender, msg.value);
}
```

ETH is occasionally transferred from the `WithdrawalVault` to `Goldsand` via `withdrawETHToGoldsand`, which works via a low-level call which doesn't include a function selector. This call gets routed to the `fallback` or `receive` method where `fund` is called. This is unexpected as transfers from `WithdrawalVault` are not intended to be used for funding.

As a result, we increment `funderToBalance[withdrawalVaultAddress]` by the amount transferred. Since we can only decrement `funderToBalance` values via `operatorWithdrawETHForUser`, and that only decrements the amount for one account, we either can't clear the amount of the actual user to withdraw for or `funderToBalance[withdrawalVaultAddress]`.

Recommendation: Include an exception in `fallback` and `receive` which doesn't call `fund` if the `msg.sender` is `withdrawalVaultAddress`:

```
  /**
   * @notice Fallback function
   * @dev Handles Ether sent directly to the contract.
   */
  fallback() external payable whenNotPaused {
+     if (msg.sender == withdrawalVaultAddress) return;
      fund();
  }

  /**
   * @notice Receive function
   * @dev Handles Ether sent directly to the contract.
   */
  receive() external payable whenNotPaused {
+     if (msg.sender == withdrawalVaultAddress) return;
      fund();
  }
```

Inshallah: Fixed in commit 43543c93.

Cantina Managed: This has been fixed by no longer withdrawing from `WithdrawalVault` to `Goldsand`.

3.2.3 `emergencyWithdraw()` cannot withdraw ETH in `WithdrawalVault`

Severity: Low Risk

Context: Goldsand.sol#L290

Description: On pause, `EMERGENCY_ROLE` can call `emergencyWithdraw()` to withdraw the ETH in Goldsand, and since `WithdrawalVault` is used to receive the ETH withdrawn by the validator, `WithdrawalVault` also holds ETH.

The problem is that `callWithdrawETH()` can only withdraw ETH from `WithdrawalVault` when it is unpaused, making it impossible for the protocol to withdraw ETH from `WithdrawalVault` when it is paused.

Recommendation: It is recommended to call `WithdrawalVault.withdrawETH()` in `emergencyWithdraw()` to allow `EMERGENCY_ROLE` to withdraw ETH from `WithdrawalVault` on pause.

Inshallah: Fixed in commit 318f127d.

Cantina Managed: Fixed.

3.2.4 Recovered tokens are locked in the contract with the current implementation

Severity: Low Risk

Context: Goldsand.sol#L261-L277

Description: In `Goldsand.sol`, we have ERC20 and ERC721 token recovery functions, which recover any tokens in the `WithdrawalVault`, transferring them to `Goldsand`:

```
/**
 * @notice Calls the withdrawal vault to recover ERC20 tokens.
 * @param _token The ERC20 token to recover.
 * @param _amount The amount of the ERC20 token to recover.
 */
function callRecoverERC20(IERC20 _token, uint256 _amount) external onlyRole(OPERATOR_ROLE) {
    IWithdrawalVault(WITHDRAWAL_VAULT_ADDRESS).recoverERC20(_token, _amount);
}

/**
 * @notice Calls the withdrawal vault to recover ERC721 tokens.
 * @param _token The ERC721 token to recover.
 * @param _tokenId The ID of the ERC721 token to recover.
 */
function callRecoverERC721(IERC721 _token, uint256 _tokenId) external onlyRole(OPERATOR_ROLE) {
    IWithdrawalVault(WITHDRAWAL_VAULT_ADDRESS).recoverERC721(_token, _tokenId);
}
```

The problem with this is that `Goldsand` doesn't contain any logic to transfer these tokens out of the contract, thereby causing them to be locked in the contract with its current implementation.

Recommendation: Implement both ERC20 and ERC721 withdrawal functions in `Goldsand` which can only be executed by an authorized party.

Inshallah: Fixed in commit 09a7e669.

Cantina Managed: Implemented by including authorized token recovery functions in both `Goldsand` and `WithdrawalVault` which allow for an arbitrary recipient.

3.2.5 Mutable `WITHDRAWAL_VAULT_ADDRESS` can result in unexpected effects

Severity: Low Risk

Context: Goldsand.sol#L147-L154

Description: In `Goldsand.setWithdrawalVaultAddress`, we have the ability to update the `WITHDRAWAL_VAULT_ADDRESS`. The `WITHDRAWAL_VAULT_ADDRESS` is the designated `WithdrawalVault` contract which we set as our `withdrawalCredentials` for deposits such that when we withdraw from our validators, the ETH goes to this address. The logic in `Goldsand` allows for us to then withdraw ETH from the `WithdrawalVault`, where it can later be distributed to users and/or the protocol.

The problem with all this is that if we update the `WITHDRAWAL_VAULT_ADDRESS` after already depositing with the initial address marked in the `withdrawalCredentials` of one or more deposits, then we'll have two separate contracts which withdrawals get sent to, while only being able to accompany one of them at a time from `Goldsand`. In this case, the only way to retrieve all ETH withdrawn to both `WithdrawalVault`'s would be to switch between them to process withdrawals.

Recommendation: Assuming that the intention for including `setWithdrawalVaultAddress` is to be able to upgrade the contract logic, we can do so without changing the contract address by placing `WithdrawalVault` behind an upgradeable proxy. We can maintain a similar effect by only allowing the `owner`, `Goldsand`, to be able to update the `implementation`, including a function to do so in `Goldsand.sol`.

Inshallah: Fixed in commits f7be71d4 and a9bc04df.

Cantina Managed: Fix implemented as recommended.

3.3 Gas Optimization

3.3.1 Redundant constructor parameter

Severity: Gas Optimization

Context: WithdrawalVault.sol#L16-L18

Description: In the `WithdrawalVault` constructor, we have two parameters: `initialOwner` and `goldsand`. However, since the `Goldsand` contract is intended to be the `owner`, both of these parameters will be the same address.

Recommendation: Remove the `initialOwner` parameter, and provide `goldsand` to `Ownable`:

```diff
- constructor(address initialOwner, IGoldsand goldsand) Ownable(initialOwner) {
+ constructor(address goldsand) Ownable(goldsand) {
-     GOLDSAND = goldsand;
+     GOLDSAND = IGoldsand(goldsand);
  }
```

Inshallah: Fixed in commits f7be71d4 and a9bc04df.

Cantina Managed: Fixed.

3.3.2 Redundant storage array

Severity: Gas Optimization

Context: Goldsand.sol#L47

Description: In `Goldsand.sol`, we have a `funders` storage array which gets pushed to every time a new user calls `fund`. This array isn't used for anything other than to output the array length in `getFundersLength`. We can save about 22100 gas for each `fund` call from a new user if we instead track this array off-chain via indexing `Funded` events.

Alternatively, if it's necessary to track the total number of funders onchain to be referenced by another contract in the future, we can instead include a `uint256` state variable which gets incremented for each new user, which would only cost 2900 gas per new user.

Recommendation: Depending on requirements, either remove the `funders` array altogether or track the total number of `funders` using a `uint256` state variable as described above.

Inshallah: Fixed in commit cf5dc6ff.

Cantina Managed: Fixed as recommended.

3.3.3 Redundant `balanceOf` function

Severity: Gas Optimization

Context: WithdrawalVault.sol#L72-L74

Description: In `WithdrawalVault.sol`, we have a `balanceOf` function which simply returns the ETH balance of the `WithdrawalVault` contract. This is redundant as instead of calling this function, a contract can simply retrieve the balance with `address(WITHDRAWAL_VAULT_ADDRESS).balance`. We can save gas on deployment by removing this function and thereby reducing the bytecode to be deployed.

Recommendation: Remove the `balanceOf` function and retrieve the balance with `address(WITHDRAWAL_VAULT_ADDRESS).balance` wherever necessary.

Inshallah: Fixed in commit cf5dc6ff.

Cantina Managed: Fixed as recommended.

3.3.4 Use constants and immutables where relevant

Severity: Gas Optimization

Context: Goldsand.sol#L71, WithdrawalVault.sol#L14

Description: If a state variable cannot be changed by any logic in the smart contract, i.e. it's effectively immutable, we can save gas by moving it from storage to a `constant` or `immutable`. In the codebase, there are two notable circumstances in which this can be done.

In `Goldsand.sol`, we initialize `DEPOSIT_CONTRACT_ADDRESS` in the `initialize` function. Since we can only execute `initialize` once, and there is no other logic to update `DEPOSIT_CONTRACT_ADDRESS`, it's effectively immutable. Instead of initializing the value in `initialize`, we can mark `DEPOSIT_CONTRACT_ADDRESS` as `immutable` and set it in the `constructor` of the implementation contract. This will save nearly `2100 + (numberOfDeposits - 1) * 100` gas by removing `SLOAD`'s in `depositFundsIfPossible`. Note that this is safe to do with the proxy as immutables in implementation contracts are correctly referenced, even via proxy contracts.

In addition to this, in `WithdrawalVault`, we initialize `GOLDSAND` in the constructor and have no further logic to be able to update it. In this case, we can simply mark `GOLDSAND` as `immutable` to save nearly 2100 gas on each `WithdrawalVault` function.

Recommendation: We can mark the above state variables as `immutable`, taking care to ensure they're initialized in the `constructor`.

- `DEPOSIT_CONTRACT_ADDRESS`:

```
- address payable private DEPOSIT_CONTRACT_ADDRESS;
+ address payable private immutable DEPOSIT_CONTRACT_ADDRESS;
```

```
- function initialize(address payable depositContractAddress) public initializer {
+ function initialize() public initializer {
      __AccessControl_init();
      __Pausable_init();
      __UUPSUpgradeable_init();
      _grantRole(DEFAULT_ADMIN_ROLE, msg.sender);
-     DEPOSIT_CONTRACT_ADDRESS = depositContractAddress;
      minEthDeposit = 0.05 ether;
  }
```

```
  constructor(address payable depositContractAddress) {
+     DEPOSIT_CONTRACT_ADDRESS = depositContractAddress;
      _disableInitializers();
  }
```

- `GOLDSAND`:

```
- IGoldsand public GOLDSAND;
+ IGoldsand public immutable GOLDSAND;
```

Inshallah: Acknowledged. In case of a large Ethereum network upgrade, we don't want to make depositContractAddress immutable, just to be on the safe side.

Cantina Managed: Acknowledged.

3.3.5 `clearDepositDatas()` should be external

Severity: Gas Optimization

Context: Goldsand.sol#L314

Description/Recommendation: `clearDepositDatas()` should be external to save gas.

Inshallah: Fixed in commit 96b61cc1.

Cantina Managed: Fixed.

3.4 Informational

3.4.1 Staking rewards cannot be computed onchain

Severity: Informational

Context: Goldsand.sol#L338

Description: In `Goldsand.operatorWithdrawETHForUser`, we decrement the `_amount` to transfer from `funderToBalance[_user]`:

```
funderToBalance[_user] -= _amount;
```

The problem with this is that the amount being withdrawn and decremented from `funderToBalance[_user]` is at most the initial deposit amount, and doesn't contain any interest accrued from staking. As a result, the `_user` cannot later be distributed rewards assuming that the rewards will be computed based on `funderToBalance` values.

In general, since the system doesn't yet accompany reward accounting or user withdrawals, it's unclear as to how rewards intend to be distributed, thus this may not be a concern if they are to be computed offchain. However, it's important to note that due to this issue, rewards cannot be safely computed onchain for any users that are withdrawn via `operatorWithdrawETHForUser`.

Furthermore, since we don't track the timing of deposits onchain, rewards cannot be computed onchain regardless because if they are computed without considering the duration that a funder has been staking then an attacker can deposit immediately before rewards are computed or distributed such that they get an amount of rewards equivalent to their relative deposit size, even though their deposit could be for only a very short duration.

Recommendation: Carefully consider how rewards will be computed and distributed. If they are to be computed offchain, consider in advance a mechanism for doing so, e.g. indexing `Funded` events based on the user and timing, and additionally a mechanism for distributing the rewards, e.g. a `MerkleDistributor`. If they are to be computed onchain, track the amount deposited per time deposited such that distribution will be relative to the amount and duration of deposits.

Inshallah: We are doing rewards offchain, we added indexed to the Funded event as suggested. We will be handling the rewards manually with the multisig until the next milestone so we don't need any mechanism like `MerkleDistributor`.

Cantina Managed: Acknowledged.

3.4.2 Unable to recover ERC1155 tokens in `WithdrawalVault`

Severity: Informational

Context: Goldsand.sol#L277

Description: Goldsand implements `callRecoverERC20()` and `callRecoverERC721()` to recover ERC20 and ERC721 tokens in `WithdrawalVault`, but does not implement a function to recover ERC1155 tokens in `WithdrawalVault`.

Recommendation: It is recommended to implement a function to recover ERC1155 tokens in `WithdrawalVault`.

Inshallah: Fixed in commit f7be71d4.

Cantina Managed: Fixed.

3.4.3 Use `mixedCase` for state variables

Severity: Informational

Context: Goldsand.sol#L76

Description: In `Goldsand.sol`, we have the state variable `WITHDRAWAL_VAULT_ADDRESS`. In line with the Solidity style guide, it's recommended that we instead use `mixedCase`, i.e.: `withdrawalVaultAddress`. This improves readability and maintainability by having the casing identify that the variable is not constant or immutable.

Recommendation: Rename `WITHDRAWAL_VAULT_ADDRESS` to `withdrawalVaultAddress`.

Inshallah: Fixed in commit cf5dc6ff.

Cantina Managed: Fixed as recommended.

3.4.4 `WithdrawalVault.initialize()` should call `__UUPSUpgradeable_init()`

Severity: Informational

Context: WithdrawalVault.sol#L41-L43

Description/Recommendation: `WithdrawalVault` inherits `UUPSUpgradeable`, so `__UUPSUpgradeable_init()` should be called in `WithdrawalVault.initialize()`.

Inshallah: Fixed in commit 9437b2f3.

Cantina Managed: Fixed.

3.4.5 Documentation errors

Severity: Informational

Context: Goldsand.sol#L189-L191, Goldsand.sol#L203-L205, Goldsand.sol#L98-L100

Description/Recommendation:

1. `fund()`/`fundOnBehalf()` no longer calls `DepositContract`:

```
      /**
      * @notice Deposits funds into the contract.
-      * @dev If we've accumulated >32 ETH and have deposit datas, we'll call
-      * the deposit contract as well.
      */
      function fund() public payable whenNotPaused {
  // ...

      /**
      * @notice Deposits funds into the contract but assigns the funds to
      * another account. This function can only be used by Goldsand.
-      * @dev If we've accumulated >32 ETH and have deposit datas, we'll call
-      * the deposit contract as well.
      */
      function fundOnBehalf(address _funderAccount) public payable onlyRole(OPERATOR_ROLE) whenNotPaused {
```

2. `depositDatas` will also be popped in `popDepositData()`:

```
- @dev This is pushed in addDepositData(...) and popped in depositFundsIfPossible().
+ @dev This is pushed in addDepositData(...) and popped in depositFundsIfPossible() and popDepositData().
```

Inshallah: Fixed in commit 96b61cc1.

Cantina Managed: Fixed.