Veridise

Auditing Report

Hardening Blockchain Security with Formal Methods

FOR

Edgeless

Edgeless Contracts

Veridise Inc.
March 22, 2024

► **Prepared For:**

Edgeless Labs
https://www.edgeless.network

► **Prepared By:**

Benjamin Sepanski
Ian Neal

► **Contact Us:** contact@veridise.com

► **Version History:**

Mar. 2, 2024	V1
Mar. 19, 2024	Initial Draft

Contents

Executive Summary 1

From Mar. 15, 2024 to Mar. 19, 2024, Edgeless Labs engaged Veridise to review the security of their Edgeless Contracts. The review covered the smart contracts used to stake into the Edgeless Network from Ethereum. Veridise conducted the assessment over 6 person-days, with 2 engineers reviewing code over 3 days on commit `e185095b`. The auditing strategy involved a tool-assisted analysis of the source code performed by Veridise engineers as well as extensive manual auditing.

Project summary. The provided source code defines a contract named `EdgelessDeposit`. Users may stake Ethereum into the deposit contract. A manager contract deposits these funds into a strategy. Currently, the only supported strategy either holds the funds, or deposits them into LIDO*. The management of these funds and strategies is performed by a whitelisted entity.

Code assessment. The Edgeless Contracts developers provided the source code of their Edgeless Contracts contracts for review. The source code appears to be original code written by the developers. It contains some documentation in the form of READMEs, which included a description of the design and listed key invariants, and documentation comments on functions and storage variables. To facilitate the Veridise auditors' understanding of the code, Edgeless Labs also provided application documentation†.

The source code contained a test suite, which the Veridise auditors noted had high coverage, testing over 95% of the lines of code in the project. Most main workflows were tested, and tests were performed within the deployment environment.

Summary of issues detected. The audit uncovered 24 issues, 2 of which are assessed to be of high or critical severity by the Veridise auditors. Specifically, V-EDG-VUL-001 describes how recovering funds after a strategy is removed requires a full contract shutdown, and V-EDG-VUL-002 describes improper tracking of Eth staked in LIDO while in the withdrawal queue. The Veridise auditors also identified 2 medium-severity issues, including emitting incorrect values (V-EDG-VUL-003) and accidentally pausing the contract when removing strategies (V-EDG-VUL-004). The auditors reported 3 low-severity issues, as well as 9 warnings and 8 informational findings. Of the 24 issues, Edgeless Labs has fixed 19 issues. This includes the 2 high-severity issues. Of the remaining issues, Edgeless Labs has acknowledged 5 issues but deemed them too minor to fix.

Recommendations. After auditing the protocol, the auditors had a few suggestions to improve the Edgeless Contracts.

* `https://lido.fi`
† `https://docs.edgeless.network`

First, the Veridise team recommends relying more heavily on mappings than arrays. While the array design is convenient for computing the total assets under management, it was also the source of several issues, such as V-EDG-VUL-001 and V-EDG-VUL-004.

Second, the Veridise team strongly recommends having a timelocked contract as the protocol owner. As described in V-EDG-VUL-006, the protocol owner can cause severe damage to the protocol, or even remove user funds. This effect is amplified by the fact that user funds may be staked into LIDO, meaning they cannot be withdrawn quickly. As mentioned in V-EDG-VUL-006, only the owner can claim LIDO withdrawals.

This leads into our third recommendation, which is to allow another mechanism by which users can cause LIDO withdrawals to be requested/claimed to further reduce this risk of centralization.

Disclaimer. We hope that this report is informative but provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the system is secure in all dimensions. In no event shall Veridise or any of its employees be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with the results reported here.

Project Dashboard 2

Table 2.1: Application Summary.

Name	Version	Type	Platform
Edgeless Contracts	`e185095b`	Solidity	Arbitrum, Ethereum

Table 2.2: Engagement Summary.

Dates	Method	Consultants Engaged	Level of Effort
Mar. 15 - Mar. 19, 2024	Manual & Tools	2	6 person-days

Table 2.3: Vulnerability Summary.

Name	Number	Fixed	Acknowledged
Critical-Severity Issues	0	0	0
High-Severity Issues	2	2	2
Medium-Severity Issues	2	2	2
Low-Severity Issues	3	0	3
Warning-Severity Issues	9	8	9
Informational-Severity Issues	8	7	8
TOTAL	24	19	24

Table 2.4: Category Breakdown.

Name	Number
Maintainability	8
Data Validation	7
Logic Error	3
Missing/Incorrect Events	3
Access Control	1
Usability Issue	1
Gas Optimization	1

3 Audit Goals and Scope

3.1 Audit Goals

The engagement was scoped to provide a security assessment of Edgeless Contracts's smart contracts. In our audit, we sought to answer questions such as:

- Are common Solidity vulnerabilities such as reentrancy, front-running, or flash loans present in the system?
- Does the wrapped token properly correspond to the amount of its underlying asset?
- Are funds properly tracked and accounted for?
- Can funds become locked in the protocol?
- Can funds be stolen from the protocol?
- Are upgradeability best practices followed?
- What authority are centralized entities given over user funds?

3.2 Audit Methodology & Scope

Audit Methodology. To address the questions above, our audit involved a combination of human experts and automated program analysis & testing tools. In particular, we conducted our audit with the aid of the following techniques:

- *Static analysis.* To identify potential common vulnerabilities, we leveraged our custom smart contract analysis tool Vanguard. These tools are designed to find instances of common smart contract vulnerabilities, such as reentrancy and uninitialized variables.
- *Fuzzing/Property-based Testing.* We also leverage fuzz testing to determine if the protocol may deviate from the expected behavior. To do this, we formalize the desired behavior of the protocol as [V] specifications and then use our fuzzing framework OrCa to determine if a violation of the specification can be found. See Section 5 for more information.

Scope. The scope of this audit is limited to the `src/` folder of the source code provided by the Edgeless Contracts developers, which contains the smart contract implementation of the Edgeless Contracts. In particular, the following contracts within the `src/` were in scope:

- `Constants.sol`
- `WrappedToken.sol`
- `EdgelessDeposit.sol`
- `StakingManager.sol`
- `strategies/EthStrategy.sol`

Methodology. Veridise auditors reviewed the reports of previous audits for Edgeless Contracts, inspected the provided tests, and read the Edgeless Contracts documentation. They then began a manual audit of the code assisted by both static analyzers and automated testing. During the audit, the Veridise auditors regularly met with the Edgeless Contracts developers to ask questions about the code.

3.3 Classification of Vulnerabilities

When Veridise auditors discover a possible security vulnerability, they must estimate its severity by weighing its potential impact against the likelihood that a problem will arise. Table 3.1 shows how our auditors weigh this information to estimate the severity of a given issue.

Table 3.1: Severity Breakdown.

	Somewhat Bad	Bad	Very Bad	Protocol Breaking
Not Likely	Info	Warning	Low	Medium
Likely	Warning	Low	Medium	High
Very Likely	Low	Medium	High	Critical

In this case, we judge the likelihood of a vulnerability as follows in Table 3.2:

Table 3.2: Likelihood Breakdown

Not Likely	A small set of users must make a specific mistake
Likely	Requires a complex series of steps by almost any user(s) - OR - Requires a small set of users to perform an action
Very Likely	Can be easily performed by almost anyone

In addition, we judge the impact of a vulnerability as follows in Table 3.3:

Table 3.3: Impact Breakdown

Somewhat Bad	Inconveniences a small number of users and can be fixed by the user
Bad	Affects a large number of people and can be fixed by the user - OR - Affects a very small number of people and requires aid to fix
Very Bad	Affects a large number of people and requires aid to fix - OR - Disrupts the intended behavior of the protocol for a small group of users through no fault of their own
Protocol Breaking	Disrupts the intended behavior of the protocol for a large group of users through no fault of their own

Vulnerability Report 4

In this section, we describe the vulnerabilities found during our audit. For each issue found, we log the type of the issue, its severity, location in the code base, and its current status (i.e., acknowledged, fixed, etc.). Table 4.1 summarizes the issues discovered:

Table 4.1: Summary of Discovered Vulnerabilities.

ID	Description	Severity	Status
V-EDG-VUL-001	Fund recovery after strategy removal requires f. . .	High	Fixed
V-EDG-VUL-002	unstEth value not tracked	High	Fixed
V-EDG-VUL-003	Wrong amount emitted on withdraw()	Medium	Fixed
V-EDG-VUL-004	Removing strategy may pause contract	Medium	Fixed
V-EDG-VUL-005	Missing address zero-checks	Low	Acknowledged
V-EDG-VUL-006	Centralization Risk	Low	Acknowledged
V-EDG-VUL-007	Strategy with staked funds can be removed	Low	Acknowledged
V-EDG-VUL-008	Unused program constructs	Warning	Fixed
V-EDG-VUL-009	Initializable best practices	Warning	Fixed
V-EDG-VUL-010	Wrong amount emitted on stake	Warning	Fixed
V-EDG-VUL-011	No range check on active strategy	Warning	Fixed
V-EDG-VUL-012	Strategies may be duplicates	Warning	Fixed
V-EDG-VUL-013	Unusable strategies may be added	Warning	Fixed
V-EDG-VUL-014	Staking manager withdrawal amount unchecked	Warning	Fixed
V-EDG-VUL-015	Incorrect Lido interface function	Warning	Fixed
V-EDG-VUL-016	Issues from previous audits	Warning	Acknowledged
V-EDG-VUL-017	Unchecked approve	Info	Fixed
V-EDG-VUL-018	Variable could be immutable	Info	Fixed
V-EDG-VUL-019	override keyword unused	Info	Fixed
V-EDG-VUL-020	Code duplication	Info	Fixed
V-EDG-VUL-021	Make chain-specific values immutable	Info	Acknowledged
V-EDG-VUL-022	Typos and incorrect comments	Info	Fixed
V-EDG-VUL-023	Unused submit return value	Info	Fixed
V-EDG-VUL-024	Compiler warning in ForceCompile.sol	Info	Fixed

4.1 Detailed Description of Issues

4.1.1 V-EDG-VUL-001: Fund recovery after strategy removal requires full shutdown

Severity	High	**Commit**	e185095
Type	Logic Error	**Status**	Fixed
File(s)	src/StakingManager.sol		
Location(s)	_withdrawEth()		
Confirmed Fix At	N/A		

A `StakingManager` supports multiple strategies for a single asset. For each asset, at most one strategy is "active" at a time. When funds are staked, they are deposited into the active strategy. Note that the `StakingManager` may have funds invested in inactive strategies. They are only "inactive" because new funds are not currently being directed towards them.

When the `StakingManager.owner` wishes to withdraw funds from the protocol, they use the `withdraw()` function, which calls `_withdrawEth()` (shown in the below code snippet).

```
function _withdrawEth(uint256 amount) internal {
    IStakingStrategy strategy = getActiveStrategy(ETH_ADDRESS);
    uint256 withdrawnAmount;
    if (address(strategy) != address(0)) {
        withdrawnAmount = strategy.withdraw(amount);
    } else {
        withdrawnAmount = amount > address(this).balance ? address(this).balance :
    amount;
    }
    (bool success, bytes memory data) = staker.call{ value: withdrawnAmount }("");
    if (!success) revert TransferFailed(data);
    emit Withdraw(ETH_ADDRESS, amount);
}
```

Snippet 4.1: Definition of `_withdrawEth()`

Note that in the above function, if `ETH_ADDRESS` has an active strategy, then up to `amount` Eth is withdrawn from the `strategy`. In particular, if a strategy is active, then none of the contracts Eth balance will be touched. The only funds withdrawn will be from the active strategy. So, the second branch is only taken if `address(0)` is added as a strategy and the active strategy is set to `address(0)`.

However, once `address(0)` is added as a strategy, it *cannot be removed*, since `removeStrategy()` calls `strategy.withdraw()`. Further, `getAssetTotal()` *will always revert*, since it calls `strategy.underlyingAssetAmount()`. So, once `address(0)` is made the active strategy, core functionalities of the contract will stop operating.

Impact Whenever a strategy is removed, its invested funds are withdrawn to the `StakingManager` contract. Since `stake()` only stakes the funds sent to the `stake()` function, these withdrawn funds cannot be reinvested. Further, as described above, they cannot be withdrawn without pausing the contract. This means that, once a strategy is removed, funds invested into that strategy are locked until `address(0)` is added as a strategy, preventing the contract from operating for a particular asset.

In addition, `StakingManager` implements `receive`, so any user can send funds directly to the contract. If a user does so, mistakenly believing their funds will be staked, those funds will be locked until the contract is paused.

Recommendation Only permit allowed strategies to send Eth using the `receive()` function.

Provide a mechanism for the `StakingManager.owner` to withdraw funds from the contract balance directly.

Developer Response `removeStrategy` now no longer withdraws from the strategy before removing it. Instead, the owner may call a new function, `withdrawToStaker`, before removing the strategy which withdraws a specified amount from the strategy and sends it to the staker.

Veridise Response It is now possible to remove a strategy without withdrawing funds from it, which could cause funds to be misplaced (as the owner will need to call `ownerWithdraw` to retrieve funds from a removed strategy). This could be an issue if, for example, someone stakes just after an owner removes funds from the active strategy, but before they remove the strategy or deactivate it.

Additionally, the issue with `receive` remains unaddressed.

Developer Response A new function, `withdrawAndRemoveStrategy`, has been created to handle the case when the owner wants to withdraw funds before removing a strategy.

The `receive` function now checks that `msg.sender` is a strategy in `strategies[ETH_ADDRESS]`.

4.1.2 V-EDG-VUL-002: unstEth value not tracked

Severity	High	**Commit**	e185095
Type	Logic Error	**Status**	Fixed
File(s)	`src/strategies/EthStrategy.sol`		
Location(s)	underlyingAssetAmount()		
Confirmed Fix At	N/A		

The `EthStrategy` contract keeps funds in one of two places:

- As part of the contract balance.
- Staked into `LIDO`.

When computing the underlying asset amount, only the stEth and Eth balances are consulted.

```
function underlyingAssetAmount() external view returns (uint256) {
    return address(this).balance + LIDO.balanceOf(address(this));
}
```

Snippet 4.2: Definition of `underlyingAssetAmount()`.

However, the `owner` of the `EthStrategy` may initiate a withdrawal from LIDO. Once a withdrawal has been initiated, the `LIDO.balance` of the contract will decrease. Until the withdrawal has been finalized and claimed, the contract will not receive the corresponding Eth. For more information on the LIDO withdrawal process, see their withdrawal documentation.

Impact Several possibilities may arise:

- While funds are in the withdrawal queue, the `EdgelessDeposit` contract's `wrappedEth` token will have a larger `totalSupply()` than `stakingManager.getAssetTotal(stakingManager.ETH_ADDRESS())`. This will prevent usage of `mintEthBasedOnStakedAmount()` and lead to possible errors for protocols built on top of this system.
- `StakingManager.getAssetTotal()` will have a dramatic drop when large withdrawals are initiated. This may affect users of the protocol and cause confusion or panic, leading them to withdraw funds.
- The `EthStrategy.underlyingAssetAmount()` may return 0 even when large amounts of funds are locked in the account. Programs or managers using this function to determine when to remove a strategy from the `StakingManager` may accidentally remove a strategy when lots of funds are still stored therein. See also V-EDG-VUL-007.

Recommendation Record the amount of `amountOfStEth` recorded in each successful withdrawal request, and add that total to the result of `underlyingAssetAmount()`.

Developer Response The developer has implemented the recommended fix by tracking an `ethUnderWithdrawal` variable.

4.1.3 V-EDG-VUL-003: Wrong amount emitted on withdraw()

Severity	Medium	**Commit**	e185095
Type	Missing/Incorrect Eve	**Status**	Fixed
File(s)	`src/StakingManager.sol`		
Location(s)	_withdrawEth()		
Confirmed Fix At	N/A		

The `StakingManager`'s `withdraw()` function allows the `staker`-address to withdraw Eth from the `StakingManager` contract. If the requested `amount` to withdraw is less than what the contract has available, a smaller amount is withdrawn. The `_withdrawEth()` function implements this logic, shown in the below code snippet.

```
function _withdrawEth(uint256 amount) internal {
    IStakingStrategy strategy = getActiveStrategy(ETH_ADDRESS);
    uint256 withdrawnAmount;
    if (address(strategy) != address(0)) {
        withdrawnAmount = strategy.withdraw(amount);
    } else {
        withdrawnAmount = amount > address(this).balance ? address(this).balance : amount;
    }
    (bool success, bytes memory data) = staker.call{ value: withdrawnAmount }("");
    if (!success) revert TransferFailed(data);
    emit Withdraw(ETH_ADDRESS, amount);
}
```

Snippet 4.3: Definition of `StakingManager._withdrawEth()`

Note that the `emit Withdraw(ETH_ADDRESS, amount)` statement uses `amount` instead of `withdrawnAmount` . Since `withdrawnAmount` may be less than `amount`, even to the point of being zero, this event may mislead off-chain applications.

Impact Off-chain applications may incorrectly record withdrawn amounts. For example, `EthStrategy.withdraw(amount)` may withdraw less than `amount` Eth if the funds are locked into Lido. In this (very likely) case, the `Withdraw` event would be emitted with an `amount` which is much larger than `withdrawnAmount`.

Recommendation Use `withdrawnAmount` instead of `amount` when emitting `Withdraw`.

Developer Response The developer has implemented the recommended change.

4.1.4 V-EDG-VUL-004: Removing strategy may pause contract

Severity	Medium	**Commit**	e185095
Type	Logic Error	**Status**	Fixed
File(s)	src/StakingManager.sol		
Location(s)	removeStrategy()		
Confirmed Fix At	N/A		

The `StakingManager` contract tracks several strategies for each `asset`, stored in an array. Each `asset` with at least one strategy has an active strategy, recorded via its index.

```
mapping(address => IStakingStrategy[]) public strategies;
mapping(address => uint256) public activeStrategyIndex;
```

Snippet 4.4: Fields used to track `strategies` and the active strategy.

When removing a strategy, the manager swaps the strategy to the end of its asset's `strategies` array, and then pops from the array. If that strategy was active, the strategy now at index 0 is activated. The `removeStrategy()` function implements this logic, shown below.

```
function removeStrategy(address asset, uint256 index) external onlyOwner {
    IStakingStrategy strategy = strategies[asset][index];
    uint256 withdrawnAmount = strategy.withdraw(strategy.underlyingAssetAmount());
    uint256 lastIndex = strategies[asset].length - 1;
    strategies[asset][index] = strategies[asset][lastIndex];
    strategies[asset].pop();
    if (activeStrategyIndex[asset] == index) activeStrategyIndex[asset] = 0;
    emit RemoveStrategy(asset, strategy, withdrawnAmount);
}
```

Snippet 4.5: Definition of `removeStrategy()`

Note, however, that if `index != lastIndex`, and `lastIndex` is the currently active strategy, then the active strategy index will become invalidated. For example, consider the below scenario:

1. Three strategies are present for an `asset`.
2. The `StakingManager` sets the last strategy (index 2) to active.
3. The `StakingManager` removes the strategy at index 0.

Now, the strategy which *was* at index 2 has been swapped to index 0, but the `activeStrategyIndex` was not updated.

We implemented the above scenario as a test using the `forge` setup.

```
function _addStrategy() internal {
    vm.startPrank(owner);

    address ethStakingStrategyImpl = address(new EthStrategy());
    bytes memory ethStakingStrategyData = abi.encodeCall(EthStrategy.initialize, (
    owner, address(stakingManager)));

    IStakingStrategy newStrategy = IStakingStrategy(payable(address(new ERC1967Proxy(
    ethStakingStrategyImpl, ethStakingStrategyData))));

    stakingManager.addStrategy(stakingManager.ETH_ADDRESS(), newStrategy);
    vm.stopPrank();
}

function test_removeStrategy() external {
    _addStrategy();
    _addStrategy();
    address ETH_ADDRESS = stakingManager.ETH_ADDRESS();
    // Staking manager now has 3 Eth strategies
    for(uint i = 0; i < 3; ++i) {
        require(address(stakingManager.strategies(ETH_ADDRESS, i)) != address(0));
    }
    vm.expectRevert(); // out-of-bounds revert on access to index 3
    address(stakingManager.strategies(ETH_ADDRESS, 3)) == address(0);

    // Use 3rd Eth Strategy
    hoax(owner);
    stakingManager.setActiveStrategy(ETH_ADDRESS, 2);
    IStakingStrategy activeStrategy = stakingManager.getActiveStrategy(ETH_ADDRESS);

    // Remove first eth strategy
    hoax(owner);
    stakingManager.removeStrategy(ETH_ADDRESS, 0);

    // Active index is now invalid, even though the active strategy is still present
    require(stakingManager.strategies(ETH_ADDRESS, 0) == activeStrategy);

    // This fails! The returned active strategy is the 0-address
    require(stakingManager.getActiveStrategy(ETH_ADDRESS) == activeStrategy);
}
```

Impact When removing strategies, the protocol may obtain an invalid active strategy index. This will effectively pause the protocol until the owner sets the strategy to a valid one.

Recommendation Update the `activeStrategyIndex` when it is set to `lastIndex`. Further, we would recommend requiring the owner to provide a default value, rather than automatically defaulting to whichever strategy is at index 0.

Developer Response A new parameter has been added to `removeStrategy`, `newActiveStrategyIndex`, which sets the new active strategy if `activeStrategyIndex[asset] == index`.

Veridise Response The case where `activeStrategyIndex[assert] == lastIndex && index != lastIndex` is still problematic, as `activeStrategyIndex[assert]` will not be updated and refer to a now-invalid index.

Developer Response `activeStrategyIndex[asset]` is now unconditionally set to `newActiveStrategyIndex`.

Veridise Response There is no range check on `newActiveStrategyIndex`, so it could still be accidentally set to an invalid index by the owner. Additionally, since the new strategy index is now always updated, it might be beneficial to modify the `RemoveStrategy` event to include the `newActiveStrategyIndex`.

Developer Response We addressed the above comments.

4.1.5 V-EDG-VUL-005: Missing address zero-checks

Severity	Low	**Commit**	e185095
Type	Data Validation	**Status**	Acknowledged
File(s)	See issue description		
Location(s)	See issue description		
Confirmed Fix At	N/A		

Description The following functions take addresses as arguments, but do not validate that the addresses are non-zero:

- src/StakingManager.sol
  - `addStrategy()`: `strategy` is not validated.
- src/WrappedToken.sol
  - `constructor()`: `minter` is not validated.
- src/strategies/EthStrategy.sol
  - `initialize()`: `_stakingManager` is not validated.
  - `setStakingManager()`: `_stakingManager` is not validated.

Note that `StakingManager.setStaker` does not perform zero-checks. However, this is currently necessary to prevent locked funds. See V-EDG-VUL-001 for more details.

Impact As described in V-EDG-VUL-001, the `owner` may pass `address(0)` as an attempt to recover funds. However, adding a zero-address `strategy` can cause core functionality of the contract to stop functioning permanently.

If zero is passed for other addresses, core functions of the contract will revert in all cases.

Note that, with the provided deployment scripts, these addresses will be non-zero. However, there is no on-chain guarantee of this fact, so changes in the deployment script could lead to errors down the line.

Recommendation Require the addresses to be non-zero.

Developer Response The developer acknowledges that zero-checks are standard, but believes they cause more complexity than the possible problems that they solve, which is why they are omitted.

4.1.6 V-EDG-VUL-006: Centralization Risk

Severity	Low	**Commit**	e185095
Type	Access Control	**Status**	Acknowledged
File(s)	See issue description		
Location(s)	See issue description		
Confirmed Fix At	N/A		

Similar to many projects, Edgeless Lab's smart contracts declare an administrator role that is given special permissions. In particular, these administrators are given the following abilities:

- `EthStrategy:`
  - The `owner` is the only user who can initiate or finalize Lido withdrawal requests.
- `StakingManager:`
  - The `owner` can set the `staker`, add strategies, and decide the active strategy. This determines where staked funds will be sent. Setting the `staker` to themselves allows withdrawal of all funds to the `owner`.
- `EdgelessDeposit`
  - The `owner` of this contract may mint `wrappedEth` tokens to whoever they choose, up to but not exceeding a 1-1 ratio with the underlying staked assets.

Impact If a private key were stolen, a hacker would have access to sensitive functionality that could compromise the project. For example, a malicious `owner` could change the `StakingManager.staker` to themselves, and withdraw all staked funds from the protocol.

Further, note that only the owner can request or claim LIDO withdrawal requests. If the owner stops operating or is compromised, those user funds may be permanently locked. The owner may also decide not to perform withdrawals in order to keep the value of staked funds high.

Recommendation As these are all particularly sensitive operations, we would encourage the developers to utilize a decentralized governance or multi-sig contract as opposed to a single account, which introduces a single point of failure.

Developer Response The developer has acknowledged there is a potential issue here, but accepts the risk as a limitation of the current system.

4.1.7 V-EDG-VUL-007: Strategy with staked funds can be removed

Severity	Low	**Commit**	e185095
Type	Data Validation	**Status**	Acknowledged
File(s)	`src/StakingManager.sol`		
Location(s)	removeStrategy()		
Confirmed Fix At	N/A		

The `StakingManager` manages several strategies for each `asset`. The `owner` can remove strategies, during which the contract attempts to withdraw funds from the strategy.

```
function removeStrategy(address asset, uint256 index) external onlyOwner {
    IStakingStrategy strategy = strategies[asset][index];
    uint256 withdrawnAmount = strategy.withdraw(strategy.underlyingAssetAmount());
    uint256 lastIndex = strategies[asset].length - 1;
    strategies[asset][index] = strategies[asset][lastIndex];
    strategies[asset].pop();
    if (activeStrategyIndex[asset] == index) activeStrategyIndex[asset] = 0;
    emit RemoveStrategy(asset, strategy, withdrawnAmount);
}
```

Snippet 4.6: Definition of `removeStrategy()`.

However, `strategy.withdraw()` is not guaranteed to withdraw the requested amount. For example, if `strategy` is an `EthStrategy`, it may have funds locked in LIDO which cannot be withdrawn immediately. Note also that, as described in V-EDG-VUL-002, the return value of `underlyingAssetAmount()` may be inaccurate.

Impact `StakingManager.owners` may accidentally or intentionally remove strategies with staked funds still in them. This can lead to insolvency, make funds much more difficult to track, and prevent withdrawals from the system.

Recommendation Resolve V-EDG-VUL-002, and require a strategy to have an underlying asset amount of zero before removal.

Developer Response The developers will take an off-chain approach to tracking removed strategies with remaining funds, as a future version of the project may have reasons to remove such strategies.

4.1.8 V-EDG-VUL-008: Unused program constructs

Severity	Warning	**Commit**	e185095
Type	Maintainability	**Status**	Fixed
File(s)	See issue description		
Location(s)	See issue description		
Confirmed Fix At	N/A		

Description The following program constructs are unused:

- Dependencies:
  - `@eth-optimism/sdk` is unused and not being maintained, so it should be removed.
- src/StakingManager.sol:
  - `address public depositor`
  - `bool public autoStake`: while this variable has a setter, no code in the project ever reads from this field.
- src/EdgelessDeposit.sol
  - `address public l2Eth`
  - `event SetL2Eth`
  - `function upgrade()`: this function is a no-op.

Impact These constructs may become out of sync with the rest of the project, leading to errors if used in the future.

Recommendation Remove the unused program constructs.

Developer Response The unused program constructs have been removed.

4.1.9 V-EDG-VUL-009: Initializable best practices

Severity	Warning	**Commit**	e185095
Type	Data Validation	**Status**	Fixed
File(s)	See issue description		
Location(s)	See issue description		
Confirmed Fix At	`https://github.com/edgelessNetwork/contracts/pull/32`		

The following contracts are `Initializable`, but do not follow all of OpenZeppelin's documented upgradeability best practices.

- `EdgelessDeposit`
- `EthStrategy`
- `StakingManager`

Specifically,

- None of the above have a `constructor()` invoking `_disableInitializers()`.
- None of the above invoke `__Ownable2StepUpgradeable_init` or `__UUPSUpgradeable_init` in their `initialize()` functions.
- `EdgelessDeposit` uses `__Ownable_init_unchained()` instead of `__Ownable_init()`.

Impact Allowing the implementation contract to be initialized may lead to potential scams if a malicious party initializes the implementation contract.

Not invoking the parent initializer functions may lead to problems in the future if dependent implementations are upgraded.

Recommendation Upgradeable contracts should

- Invoke `_disableInitializers()` in the constructor.
- Call parent initializers in the child `initializer()` contract.

Developer Response Initializer best practices have been implemented in the three listed contracts.

4.1.10 V-EDG-VUL-010: Wrong amount emitted on stake

Severity	Warning	**Commit**	e185095
Type	Missing/Incorrect Eve	**Status**	Fixed
File(s)	src/StakingManager.sol		
Location(s)	stake()		
Confirmed Fix At	N/A		

The `StakingManager`'s `stake()` function allows the `staker`-address to stake Eth into the active Eth strategy. The `stake()` function is shown below.

```
function stake(address asset, uint256 amount) external payable onlyStaker {
    _stakeEth(msg.value);
    emit Stake(asset, amount);
}
```

Snippet 4.7: Definition of `StakingManager.stake()`

Note that the `asset` and `amount` are not used for actual staking, but are emitted via the `Stake` event.

Impact Off-chain applications may incorrectly evaluate the amount staked. A buggy or malicious staker could cause `Stake(asset, amount)` to be emitted for any `asset` and any `amount` at the cost of only a few wei.

With the current deployment script, this is prevented by setting the `EdgelessDeposit` contract as the `staker` for the `StakingManager`.

Recommendation Since `stake()` only supports staking Eth, either require that `asset` is `ETH_ADDRESS` or remove the argument.

Similarly, either require that `amount` is equal to `msg.value`, or remove the argument.

Developer Response `stake` now `requires` that `assert == ETH_ADDRESS`, and `_stakeEth` is now called with `amount` instead of `msg.value`.

Veridise Response If `amount < msg.value`, only `amount` will be staked. Furthermore, if `amount > msg.value`, the balance of the staking contract itself will be reduced. This currently does not cause a problem since `EdgelessDeposit.depositEth` (the only caller of `stake`) ensures that `amount == msg.value`, but we still recommend removing the `amount` argument and directly using `msg.value`, as future code updates to `EdgelessDeposit` may cause `amount != msg.value` and cause bugs in the protocol.

Developer Response We removed the `amount` parameter and only rely on `msg.value`.

4.1.11 V-EDG-VUL-011: No range check on active strategy

Severity	Warning	**Commit**	e185095
Type	Data Validation	**Status**	Fixed
File(s)	`src/StakingManager.sol`		
Location(s)	setActiveStrategy()		
Confirmed Fix At	N/A		

The `StakingManager` tracks which strategy is active for a given asset by recording the `index` of that strategy within the `strategies` array.

When setting a new strategy to be active, no range check is performed

```
function setActiveStrategy(address asset, uint256 index) external onlyOwner {
    activeStrategyIndex[asset] = index;
    emit SetActiveStrategy(asset, index);
}
```

Snippet 4.8: Definition of `setActiveStrategy()`.

Impact Several key functions (such as `stake()` and `withdraw()`) will revert if the strategy index is out of bounds, effectively pausing the contract.

An owner may intentionally or accidentally pause the contract from operating by setting the active strategy to an invalid index.

Recommendation Require the `index` to be a valid index for the `activeStrategyIndex[asset]` array.

Developer Response A range check has been added to `setActiveStrategy`.

4.1.12 V-EDG-VUL-012: Strategies may be duplicates

Severity	Warning	**Commit**	e185095
Type	Data Validation	**Status**	Fixed
File(s)	`src/StakingManager.sol`		
Location(s)	addStrategy()		
Confirmed Fix At	N/A		

The `addStrategy()` function allows the `StakingManager.owner` to add a new strategy for an asset.

```
function addStrategy(address asset, IStakingStrategy strategy) external onlyOwner {
    strategies[asset].push(strategy);
    emit AddStrategy(asset, strategy);
}
```

Snippet 4.9: Definition of `addStrategy()`.

However, there is nothing preventing the owner from adding the same strategy more than once.

Impact An owner may accidentally or intentionally add duplicate strategies.

This may be done intentionally to inflate the result of `getAssetTotal()`. Adding the same strategy twice would double-count all of the assets within that strategy. This could allow an owner to use `EdgelessDeposit.mintEthBasedOnStakedAmount()` to mint more wrapped Eth than the amount of underlying Eth.

Note that a malicious owner may already add bogus strategies which return false values from `getAssetTotal()`, this simply supplies a more subtle way for them to do so which may be more difficult for protocol users to track.

If accidental, this may make removing strategies more difficult, as the owner must remove each instance of the duplicate strategy.

Recommendation Track which strategies have been added, and do not allow them to be added to the `StakingManager` more than once.

Developer Response `addStrategy` has been modified to check that the requested `strategy` has not already been added for a given `asset`.

Veridise Response The current implementation still allows for duplicate strategies to be across different assets. Since the interface of a strategy currently only allows it to support one asset, duplication could still occur in the future.

We recommend tracking a strategy's presence in the contract using a mapping to allow for this check.

Developer Response We applied the recommendation.

4.1.13 V-EDG-VUL-013: Unusable strategies may be added

Severity	Warning	**Commit**	e185095
Type	Usability Issue	**Status**	Fixed
File(s)	`src/StakingManager.sol`		
Location(s)	addStrategy()		
Confirmed Fix At	N/A		

The `addStrategy()` function allows the `StakingManager.owner` to add new strategies for various assets.

```
function addStrategy(address asset, IStakingStrategy strategy) external onlyOwner {
   strategies[asset].push(strategy);
   emit AddStrategy(asset, strategy);
}
```

Snippet 4.10: Definition of `addStrategy()`.

However, the `stake()` and `withdraw()` functions only allow deposits/withdrawals for the `ETH_ADDRESS` asset.

Impact Users may be confused by enabled strategies which cannot be interacted with. An owner may easily misconfigure the contract, or add strategies ahead of an upgrade to attempt to hide their presence.

Recommendation Either require that `asset` is `ETH_ADDRESS`, or remove the `asset` argument.

Developer Response `addStrategy` now requires `ETH_ADDRESS == asset`.

4.1.14 V-EDG-VUL-014: Staking manager withdrawal amount unchecked

Severity	Warning	**Commit**	e185095
Type	Data Validation	**Status**	Fixed
File(s)	`src/EdgelessDeposit.sol`		
Location(s)	withdrawEth()		
Confirmed Fix At	N/A		

As described in V-EDG-VUL-003, the `StakingManager.withdraw()` function may withdraw less than the requested amount. However, the `EdgelessDeposit.withdrawEth()` function does not check how much the `StakingManager` actually withdraws.

```
function withdrawEth(address to, uint256 amount) external {
    wrappedEth.burn(msg.sender, amount);
    stakingManager.withdraw(amount);
    (bool success, bytes memory data) = to.call{ value: amount }("");
    if (!success) revert TransferFailed(data);
    emit WithdrawEth(msg.sender, to, amount, amount);
}
```

Snippet 4.11: Definition of `EdgelessDeposit.withdrawEth()`.

Impact If future changes allow Eth to accumulate to the `EdgelessDeposit`'s balance, this may allow external actors to withdraw funds from the `EdgelessDeposit` contract instead of the `stakingManager`.

Recommendation Add a return value to `StakingManager.withdraw()`, and check that the returned value is equal to `amount`.

Developer Response The recommended return values and checks have been added.

4.1.15 V-EDG-VUL-015: Incorrect Lido interface function

Severity	Warning	**Commit**	e185095
Type	Data Validation	**Status**	Fixed
File(s)	`src/interfaces/ILido.sol`		
Location(s)	submit()		
Confirmed Fix At	N/A		

The Lido interface defines the `submit` function with the following signature:

```
function submit(address referralUser) external payable;
```

Snippet 4.12: Snippet from ILido.sol

However, the official Lido contract defines the function with a return value, which is the number of stEth shares generated:

```
/**
 * @notice Send funds to the pool with optional _referral parameter
 * @dev This function is alternative way to submit funds. Supports optional referral
    address.
 * @return Amount of StETH shares generated
 */
function submit(address _referral) external payable returns (uint256)
```

Snippet 4.13: Snippet from Lido.sol

Impact While this doesn't affect the ability for the function to be called (as this definition generates the same function selector), this definition is (1) inconsistent with the official deployed contract and (2) prevents contracts from using the return value for error messages and/or other data validation.

Recommendation We recommend adding the return value to the definition and handling it appropriately where called. Additionally, we recommend adding documentation to this interface that shows where the definition originates from.

Developer Response Both recommendations have been implemented.

4.1.16 V-EDG-VUL-016: Issues from previous audits

Severity	Warning	**Commit**	e185095
Type	Maintainability	**Status**	Acknowledged
File(s)	See issue description		
Location(s)	See issue description		
Confirmed Fix At	N/A		

Some issues from prior audits remain unresolved and without a response, including

1. Emitting extra events.
2. Unnecessarily high time complexity in hint computations.
3. Treating stEth as equal to Eth.

Impact Not having a clear explanation of why an issue was acknowledge/ignored may lead to concern about the protocol from users or make important protocol assumptions unclear.

For example, the Veridise audit team's understanding is that Edgeless treats stEth as equal to Eth because they are only exchanged through LIDO staking/unstaking, which (except in the case of a mass slashing event) processes withdrawals at a 1:1 ratio. If the `EthStrategy` were to instead trade the `stEth` on exchanges, it would need to properly account for the possibility of a depeg by querying an oracle.

We do strongly recommend the protocol developers monitor the LIDO share rate and bunker mode status to ensure solvency.

Recommendation Respond to or resolve each issue from prior reports.

Developer Response

1. We use the new events for our own internal tracking.
2. We don't believe the gas savings in the first listed issue are worth the required development changes.
3. We are aware of the potential for a depeg. Depegging is not an issue, since we would be redeeming directly for the underlying and LIDO will always allow this unless there is mass slashing, in which case we will pause and redeem pro rata.

4.1.17 V-EDG-VUL-017: Unchecked approve

Severity	Info	**Commit**	e185095
Type	Maintainability	**Status**	Fixed
File(s)	`src/strategies/EthStrategy.sol`		
Location(s)	requestLidoWithdrawal()		
Confirmed Fix At	N/A		

The `requestLidoWithdrawal()` function calls `LIDO.approve()`, but does not use the return value.

```
function requestLidoWithdrawal(uint256[] calldata amounts)
    external
    onlyOwner
    returns (uint256[] memory requestIds)
{
    uint256 total;
    for (uint256 i; i < amounts.length; ++i) {
        total += amounts[i];
    }
    LIDO.approve(address(LIDO_WITHDRAWAL_ERC721), total);
    requestIds = LIDO_WITHDRAWAL_ERC721.requestWithdrawals(amounts, address(this));
    emit RequestedLidoWithdrawals(requestIds, amounts);
}
```

Snippet 4.14: Definition of `requestLidoWithdrawal()`

Some ERC20 tokens return `false` on an `approve` instead of reverting, which is allowed in the standard.

Impact While LIDO does revert on a failed approve (see e.g. this line from Lido's implementation), this may lead to maintainability issues if other strategies are based on this code, or if this strategy is used on a fork of Lido with different approval behavior.

Recommendation `require()` that `LIDO.approve()` returns `true`.

Developer Response The developer implemented the recommended fix.

4.1.18 V-EDG-VUL-018: Variable could be immutable

Severity	Info	**Commit**	e185095
Type	Gas Optimization	**Status**	Fixed
File(s)	`src/WrappedToken.sol`		
Location(s)	address public minter		
Confirmed Fix At	N/A		

The `minter` field in `WrappedToken` is set once, and then never set again, but is not declared as `immutable`.

Impact Declaring `minter` as `immutable` would decrease gas costs and increase code clarity.

Recommendation Make the `minter` field `immutable`.

Developer Response The developer has implemented the suggested change.

4.1.19 V-EDG-VUL-019: override keyword unused

Severity	Info	**Commit**	e185095
Type	Maintainability	**Status**	Fixed
File(s)	`src/strategies/EthStrategy.sol`		
Location(s)	See issue description		
Confirmed Fix At	`https://github.com/edgelessNetwork/contracts/pull/26/`		

In the `EthStrategy` contract, the `override` keyword is not used on functions which implement `IStakingStrategy` methods.

Impact If a function is removed from an interface, contracts implementing that interface may forget to remove the corresponding function.

This can lead to code bloat or unintended functionality reaching deployment.

Recommendation Add the `override` keyword to any functions which implement an interface method.

Developer Response `override` has been added to applicable function signatures.

4.1.20 V-EDG-VUL-020: Code duplication

Severity	Info	**Commit**	e185095
Type	Maintainability	**Status**	Fixed
File(s)	`src/strategies/EthStrategy.sol`		
Location(s)	See issue description		
Confirmed Fix At	N/A		

The `EthStrategy` contract provides two `deposit()` endpoints. The first is `deposit()`, which only the `stakingManager` can call. The second is `ownerDeposit()`, which only the `owner` can call. `ownerDeposit()` always stakes funds into `LIDO`, whereas `deposit()` only stakes funds into `LIDO` if the `autoStake` flag is set to `true`. The function definitions are shown below.

```
function deposit(uint256 amount) external payable onlyStakingManager {
    if (!autoStake) return;
    if (amount > address(this).balance) revert InsufficientFunds();
    LIDO.submit{ value: amount }(address(0));
    emit EthStaked(amount);
}

function ownerDeposit(uint256 amount) external payable onlyOwner {
    if (amount > address(this).balance) revert InsufficientFunds();
    LIDO.submit{ value: amount }(address(0));
    emit EthStaked(amount);
}
```

Snippet 4.15: Definitions of `deposit` (called by the `stakingManager`) and `ownerDeposit` (called by the `owner`)

The above two functions share most of their implementation. Rather than being extracted into an internal function, the code is duplicated.

A similar duplication occurs between functions `withdraw()` and `ownerWithdraw()`.

Impact Changes made to the codebase may not be reflected in both instantiations of the deposit/withdrawal logic.

Recommendation Create internal functions to abstract away the shared logic between:

- `deposit()` and `ownerDeposit()`.
- `withdraw()` and `ownerWithdraw()`.

Developer Response The developer has implemented the recommended code refactoring.

4.1.21 V-EDG-VUL-021: Make chain-specific values immutable

Severity	Info	**Commit**	e185095
Type	Maintainability	**Status**	Acknowledged
File(s)	`src/Constants.sol`		
Location(s)	See issue description		
Confirmed Fix At	N/A		

The addresses for `LIDO` and `LIDO_WITHDRAWAL_ERC721` are hard-coded as constants.

```
ILido constant LIDO = ILido(0xae7ab96520DE3A18E5e111B5EaAb095312D7fE84);
IWithdrawalQueueERC721 constant LIDO_WITHDRAWAL_ERC721 =
    IWithdrawalQueueERC721(0x889edC2eDab5f40e902b864aD4d7AdE8E412F9B1);
```

Snippet 4.16: Snippet from `src/Constants.sol`.

However, they could instead be supplied during deployment and set to an `immutable` field.

Impact When deploying contracts on multiple chains, it is easy to forget to update and re-compile the code for each chain.

Recommendation Use `immutable` fields instead of hard-coded `constants`.

Developer Response The developer has acknowledged the comment, but does not need to make any changes as there are currently no plans to deploy on other chains.

4.1.22 V-EDG-VUL-022: Typos and incorrect comments

Severity	Info	**Commit**	e185095
Type	Maintainability	**Status**	Fixed
File(s)	See issue description		
Location(s)	See issue description		
Confirmed Fix At	N/A		

Description In the following locations, the auditors identified minor typos and potentially misleading comments:

- src/EdgelessDeposit.sol
  - `_mintWrappedEth()`: The natspec comment for this function references a non-existent field (`autobridge`) and bridging mechanism.
- src/StakingManager.sol
  - `StakingManager`: The natspec comment for this contract references an unused field (`depositor`).

Impact These minor errors may lead to future developer confusion.

Recommendation Fix the comments to match the current implementation.

Developer Response The developer has updated the comments accordingly.

4.1.23 V-EDG-VUL-023: Unused submit return value

Severity	Info	**Commit**	e185095
Type	Missing/Incorrect Eve	**Status**	Fixed
File(s)	`src/strategies/EthStrategy.sol`		
Location(s)	deposit(), ownerDeposit()		
Confirmed Fix At	N/A		

The Lido `submit` function is invoked twice, once in `deposit` and once in `ownerDeposit`.

```
LIDO.submit{ value: amount }(address(0));
```

Snippet 4.17: Snippet from `deposit()`.

As discussed in V-EDG-VUL-015, the `submit` function should return a value (the number of StEth shares created). This value could be added to the `EthStaked` event to inform event listeners of how many StEth shares were created from the given `amount`.

Impact The `EthStaked` event is currently not as informative as it could be.

Recommendation Use the return value from the `submit` function as part of the `EthStaked` event to make it more informative.

Developer Response The `sharedGenerated` return value is now emitted in the `EthStaked` event.

4.1.24 V-EDG-VUL-024: Compiler warning in ForceCompile.sol

Severity	Info	**Commit**	e185095
Type	Maintainability	**Status**	Fixed
File(s)	`src/ForceCompile.sol`		
Location(s)	See issue description		
Confirmed Fix At	N/A		

The ForceCompile.sol file generates compiler warnings since it missing a solidity version declaration.

Recommendation Fix the compiler warning by adding the following lines to the beginning of the file:

```solidity
// SPDX-License-Identifier: UNLICENSED
pragma solidity >=0.8.23;
```

Developer Response The recommended changes have been added.

Fuzz Testing 5

5.1 Methodology

Our goal was to fuzz test Edgeless Contracts to assess its correctness. We used OrCa as our fuzzer and wrote invariants—logical formulas that should hold after every transaction. We then encoded those invariants as assertions in [V].

To handle interactions with LIDO, the Veridise auditors wrote mock contracts implementing the core logic of LIDO deposits/withdrawals.

5.2 Properties Fuzzed

Table 5.1 describes the invariants we fuzz-tested. The second column describes the invariant informally in English, and the third shows the total amount of compute time spent fuzzing this property. The last column indicates the number of bugs identified when fuzzing the invariant.

The Veridise auditors devoted a total of 10 compute-hours (600 minutes) to fuzzing this protocol, identifying a total of 1 bug (V-EDG-VUL-002).

Table 5.1: Invariants Fuzzed.

Specification	Invariant	Minutes Fuzzed	Bugs Found
V-EDG-SPEC-001	Balance of ewEth bounded by Eth and stEth	600	1
V-EDG-SPEC-002	Edgeless deposit has proper access control	600	0
V-EDG-SPEC-003	Only owner can stake to LIDO without autoStak	600	0
V-EDG-SPEC-004	Wrapped Eth has proper access control	600	0

5.3 Detailed Description of Fuzzed Specifications

5.3.1 V-EDG-SPEC-001: Balance of ewEth bounded by Eth and stEth

Minutes Fuzzed	600	Bugs Found	1

Scope This specification applies to any action which may change the Eth or stEth of protocol contracts, or which may change the total supply of ewEth.

Natural Language The total number of wrapped Eth tokens distributed by the `EdgelessDeposit` contract is bounded by the amount of Eth and stEth managed by the contract.

Formal

```
vars: EdgelessDeposit edgeless, ILido lido, IStakingStrategy strategy, WrappedToken
    wrappedEth
inv: finished(
    edgeless.*,
    wrappedEth.totalSupply() <= lido.balanceOf(strategy) + balance(strategy)
)
```

```
vars: ILido lido, IStakingStrategy strategy, WrappedToken wrappedEth
inv: finished(
    lido.*,
    wrappedEth.totalSupply() <= lido.balanceOf(strategy) + balance(strategy)
)
```

```
vars: ILido lido, IStakingStrategy strategy, WrappedToken wrappedEth
inv: finished(
    strategy.*,
    wrappedEth.totalSupply() <= lido.balanceOf(strategy) + balance(strategy)
)
```

Example Requesting a Lido withdrawal reduces the stEth balance without immediately supplying Eth. This can cause the wrapped Eth token to have a larger total supply than the combined Eth and stEth balances.

This issue can be resolved by also tracking the value of unstEth owned by the protocol.

```
test: finished(EdgelessDeposit_4.depositEth(__user1__), sender = __user2__ && value =
     766878);
      finished(strategy.requestLidoWithdrawal([0, 77]), sender = __user0__)
```

5.3.2 V-EDG-SPEC-002: Edgeless deposit has proper access control

Minutes Fuzzed	600	**Bugs Found**	1

Scope This specification applies the `EdgelessDeposit` contract.

Natural Language Only the owner should be able to change the value of key parameters like the `stakingManager` or `wrappedEth`. Only the owner should be able to upgrade.

Formal

```
vars: EdgelessDeposit edgeless
inv: finished(
    edgeless.*,
    sender = edgeless.owner()
    ||
    (
        old(edgeless.stakingManager) = edgeless.stakingManager
        &&
        old(edgeless.wrappedEth) = edgeless.wrappedEth
        &&
        old(edgeless.l2Eth) = edgeless.l2Eth
    )
)
```

```
vars: EdgelessDeposit edgeless
inv: reverted(
    edgeless.upgradeToAndCall,
    sender != edgeless.owner()
)
```

5.3.3 V-EDG-SPEC-003: Only owner can stake to LIDO without autoStake

Minutes Fuzzed	600	**Bugs Found**	1

Scope This specification applies the `EthStrategy` contract.

Natural Language Only the owner should be able to change the amount of tokens staked into LIDO, unless `autoStake` is enabled.

Formal

```
vars: EthStrategy strategy, ILido lido
inv: finished(
    strategy.*,
    strategy.autoStake()
    ||
    sender = strategy.owner()
    ||
    lido.balanceOf(strategy) = old(lido.balanceOf(strategy))
)
```

5.3.4 V-EDG-SPEC-004: Wrapped Eth has proper access control

Minutes Fuzzed	600	**Bugs Found**	1

Scope This specification applies the `WrappedToken` contract.

Natural Language Only the `EdgelessDeposit` contract should be able to mint or burn tokens.

Formal

```
vars: EdgelessDeposit edgeless, WrappedToken wrappedEth
inv: reverted(
    wrappedEth.mint,
    sender != edgeless
)
```

```
vars: EdgelessDeposit edgeless, WrappedToken wrappedEth
inv: reverted(
    wrappedEth.burn,
    sender != edgeless
)
```

Glossary

flash loan A loan which must be repaid in the same transaction, typically offered at a much more affordable rate than traditional loans . 5

front-running A vulnerability in which a malicious user takes advantage of information about a transaction while it is in the mempool. 5

LIDO A liquid staking protocol. See `https://lido.fi` for more information . 1, 37

reentrancy A vulnerability in which a smart contract hands off control flow to an unknown party while in an intermediate state, allowing the external party to take advantage of the situation. 5

smart contract A self-executing contract with the terms directly written into code. Hosted on a blockchain, it automatically enforces and executes the terms of an agreement between buyer and seller. Smart contracts are transparent, tamper-proof, and eliminate the need for intermediaries, making transactions more efficient and secure.. 1, 43

Solidity The standard high-level language used to develop smart contracts on the Ethereum blockchain. See `https://docs.soliditylang.org/en/v0.8.19/` to learn more. 5