Spiko Smart Contracts

Security Assessment (Summary Report)

November 23, 2023

Prepared for:
Antoine Michon and Samuel Briole
Spiko

Prepared by: **Guillermo Larregay**

About Trail of Bits

Founded in 2012 and headquartered in New York, Trail of Bits provides technical security assessment and advisory services to some of the world's most targeted organizations. We combine high-end security research with a real-world attacker mentality to reduce risk and fortify code. With 100+ employees around the globe, we've helped secure critical software elements that support billions of end users, including Kubernetes and the Linux kernel.

We maintain an exhaustive list of publications at https://github.com/trailofbits/publications, with links to papers, presentations, public audit reports, and podcast appearances.

In recent years, Trail of Bits consultants have showcased cutting-edge research through presentations at CanSecWest, HCSS, Devcon, Empire Hacking, GrrCon, LangSec, NorthSec, the O'Reilly Security Conference, PyCon, REcon, Security BSides, and SummerCon.

We specialize in software testing and code review projects, supporting client organizations in the technology, defense, and finance industries, as well as government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, and Zoom.

Trail of Bits also operates a center of excellence with regard to blockchain security. Notable projects include audits of Algorand, Bitcoin SV, Chainlink, Compound, Ethereum 2.0, MakerDAO, Matic, Uniswap, Web3, and Zcash.

To keep up to date with our latest news and announcements, please follow @trailofbits on Twitter and explore our public repositories at https://github.com/trailofbits. To engage us directly, visit our "Contact" page at https://www.trailofbits.com/contact, or email us at info@trailofbits.com.

Trail of Bits, Inc.
228 Park Ave S #80688
New York, NY 10003
https://www.trailofbits.com
info@trailofbits.com

Notices and Remarks

Copyright and Distribution

Test Coverage Disclaimer

All activities undertaken by Trail of Bits in association with this project were performed in accordance with a statement of work and agreed upon project plan.

Security assessment projects are time-boxed and often reliant on information that may be provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

Trail of Bits uses automated testing techniques to rapidly test the controls and security properties of software. These techniques augment our manual security review work, but each has its limitations: for example, a tool may not generate a random edge case that violates a property or may not fully complete its analysis during the allotted time. Their use is also limited by the time and resource constraints of a project.

Table of Contents

Project Summary

Contact Information

The following project manager was associated with this project:

Anne Marie Barry, Project Manager
annemarie.barry@trailofbits.com

The following engineering director was associated with this project:

Josselin Feist, Engineering Director, Blockchain
josselin.feist@trailofbits.com

The following consultant was associated with this project:

Guillermo Larregay, Consultant
guillermo.larregay@trailofbits.com

Project Timeline

The significant events and milestones of the project are listed below.

Date	Event
September 28, 2023	Pre-project kickoff call
October 10, 2023	Delivery of report draft
October 10, 2023	Report readout meeting
November 23, 2023	Delivery of summary report with fix review appendix

Executive Summary

Engagement Overview

Spiko engaged Trail of Bits to review the security of the Spiko smart contracts in commit `e0607206`. During the audit, minor modifications were added in commits `bf18851` (adding ownership to the token contract) and `61bc787` (updating OZ dependencies to contracts in v5.0.0)

Spiko is a regulated financial entity in the European Union (EU). Its first financial products will be two regulated funds whose shares will be issued on EVM-compatible blockchains. The in-scope smart contracts allow investors to buy and sell fund shares using either fiat currency (Euros or US dollars) or allowed stablecoins. As a regulated entity, Spiko cannot manipulate money in any form, so it integrates with a licensed third-party asset service provider for on- and off-ramp services. Due to regulatory constraints, Spiko must keep a list of allowed addresses and publish the fund share net asset value daily using an on-chain oracle.

One consultant conducted the review from October 2 to October 6, 2023, for a total of one engineer-week of effort. With full access to source code and internal project documentation, we performed static and dynamic testing of the target codebase, using automated and manual processes.

Observations and Impact

During this review, we focused on all the contracts and tests in the repository. This includes, in priority order, the `PermissionManager` and `PermissionManaged` permission manager contracts, the `Token` and `Redemption` contracts, the `Oracle` contract, and the `ERC20Mock` and `ERC1363ReceiverMock` token mock contracts. Additionally, we reviewed the `Masks` utility library. We did not consider the dependencies, deploy/migration scripts, or any off-chain components outside of the in-scope contracts.

We identified only four informational- to low-severity findings, three of them related to data validation. One of these issues allows the oracle operator to modify the last price or lock the oracle; a second finding allows users to lock tokens in the `Redemption` contract; and the third issue is related to the minimum amount of tokens required to redeem shares. The fourth finding concerned testing procedures—specifically, some of the tests were not executing correctly.

We also performed a codebase maturity evaluation for the project, highlighting parts of the system that could be improved to increase the overall quality of the codebase. Lastly, this report includes appendices with information and recommendations to prevent the accidental inclusion of bugs in the system and details on how to respond if a security incident happens.

Recommendations

We recommend that the Spiko team do the following:

- **Remediate the findings disclosed in this report.** These findings should be addressed as part of a direct remediation or as part of any refactor that may occur when addressing other recommendations.
- **Review the mutation testing results.** Analyze the results for expected and unexpected behavior in contract functions.
- **Modify the test suite.** Fix the test suite to avoid false negative results when it is run. Add new tests for the cases not covered in the provided test suite

Fix Review Summary

The Spiko team submitted fixes for two of the issues and accepted the risks of the two remaining issues. The Spiko team provided notes for the issues that were not fixed and explained the reasoning behind their decision to not address them.

In particular, the issues with the test suite and the minimum redemption amount were fixed. Additionally, new test cases were added for the modified functionality and for the existing codebase.

Codebase Maturity Evaluation

Trail of Bits uses a traffic-light protocol to provide each client with a clear understanding of the areas in which its codebase is mature, immature, or underdeveloped. Deficiencies identified here often stem from root causes within the software development life cycle that should be addressed through standardization measures (e.g., the use of common libraries, functions, or frameworks) or training and awareness programs.

Category	Summary	Result
Arithmetic	The in-scope contracts use a recent version of the Solidity compiler, specifically versions ^0.8.20. This includes every contract except for the mock contracts, which are compiled using Solidity versions ^0.8.0. These Solidity versions implement automatic checks for arithmetic operations, which prevents issues such as underflow and overflow. The code does not contain any complex arithmetic operations or explicit unchecked blocks.	**Strong**
Auditing	The audited code is part of a system that includes both on-chain and off-chain components that communicate by emitting and listening to events. The contracts were designed to emit events in all state-changing functions. The Spiko team mentioned a regulation compliance requirement that mandates an incident monitoring and response plan. However, neither details nor documentation of this incident monitoring and response plan were available during the current audit. Therefore, it was not possible to determine the adequacy of the plan in conformance to this codebase maturity evaluation.	**Moderate**
Authentication / Access Controls	The system has a well-structured actor and role operation scheme. Privileged operations are handled through multisignature accounts and hardware wallets. The test suite includes specific tests for privileged function calls.	**Strong**

	Both the design and implementation of these operations are defined in the system's documentation. This includes the privilege groups and permissions, as well as the multisignature thresholds required for each privileged function call. Some of the roles in the system are managed through the API provided by OpenZeppelin Defender. The permissions for each actor are organized in permission groups. Group administrators are allowed to grant or revoke these permissions individually. In the `Token` contract, the ownership serves no purpose other than to comply with a suggested improvement to Uniswap to limit pool creation for permissioned tokens. All privileged functions are managed through the Spiko permissions management contracts, and the `Token` contract's "owner" has no special privileges.	
Complexity Management	The contract functions have clear, well-defined purposes. There is no redundant code, and third-party libraries are used when needed. Inheritance trees can be complex at times because most of them inherit from OpenZeppelin contracts for upgradeability, multicall support, and other required behavior.	**Satisfactory**
Decentralization	The system features a high degree of centralization by design, which is mandated by compliance requirements established by EU regulators. The Spiko team can unilaterally and immediately upgrade contracts, modify system parameters, manage the allowed users list, and perform any other privileged operation. For a new user to be allowed into the system, they must go through know-your-customer (KYC) and anti–money laundering procedures. All user accounts interacting with the system must be tied to a single individual or legal entity.	**Weak**

Documentation	The provided documentation for this audit is internal and specifically targeted to auditors, rather than the users. The documentation is well structured and features a comprehensive outline of the system, its components, and its operation. The first section introduces the project, providing a high-level overview of the system, its parts, and how they interact. It includes illustrative workflow graphics to better explain user operations. It also features an explanation of all regulatory constraints that directly impact the deployment and operation of the contracts. The next section presents the smart contract architecture and, for each contract, defines its purpose, the number of deployed instances, and a list of the expected behavior and restrictions. The last section explains the actors and roles, with all permissions for each role, and the private key management for each actor. No end user documentation was provided, and the smart contract in-code documentation is erratic: some contracts are well commented, and others are not commented at all. The documentation provided is clear and explains the entire on-chain system at a high level without the need to read code.	**Satisfactory**
Low-Level Manipulation	Few instances of low-level operations are present in the in-scope contracts: • The `Masks` library, responsible for encoding permissions and groups as a bit field in a `bytes32` variable, uses low-level binary operations such as `bitwise-AND` and `bitwise-OR`. There is no specific documentation for this library, but the uses are straightforward. • The `ERC1363Upgradeable` contract uses a few lines of uncommented assembly code to propagate revert reasons.	**Moderate**

Testing and Verification	The tests' branch and function coverage is 100% for all contracts in scope except the following: • `Token` contract: The coverage was 100% prior to the inclusion of commit `bf18851`, where new code was introduced but no new test cases were added. • `ERC20Mock`, `ERC1363ReceiverMock`, and `ERC1363Upgradeable` contracts: They are used to interact with the system, but the test suite has no cases for verifying the implementations themselves. • `Masks` library: There are functions in the library that were added for feature completeness, but they are not used (and therefore, not tested) in the codebase. Mutation testing revealed flaws in the test suite implementation that led to false negative results in some tests. This hid the fact that some of the code is not being tested as expected.	**Moderate**
Transaction Ordering	The designed system is not at risk of time dependency or order dependency of transactions. This is due to the combination of the system design and the regulatory constraints that must be complied with. The process of subscribing or redeeming shares for fiat currencies or stablecoins is fulfilled off-chain and is not processed instantaneously; it can take up to seven days. The share value used for these operations is the closing NAV of the day, which is known after the cutoff hour of the fund.	**Strong**

Summary of Findings

The table below summarizes the findings of the review, including type and severity details.

ID	Title	Type	Severity
1	Event emissions are ignored in the provided test suite	Testing	**Informational**
2	`publishPrice` can modify an existing price or lock the oracle	Data Validation	**Informational**
3	Tokens can be locked in the Redemption contract	Data Validation	**Low**
4	No minimum amount required for redemption	Data Validation	**Low**

Detailed Findings

1. Event emissions are ignored in the provided test suite	
Severity: **Informational**	Difficulty: Low
Type: Testing	Finding ID: TOB-SPIKO-1
Target: `test/main.test.js`	

Description

Mutation tests revealed that the provided test suite is not correctly checking for the emission of certain events.

In `Token.sol`, the path that reverts with the custom `UnauthorizedFrom` reason is not tested thoroughly. When replacing the condition with `if(false)` or replacing the custom revert with a normal revert, the tests still pass.

In `Oracle.sol`, the emission of the `Update` event in the `publishPrice` function is not correctly checked in the tests. In addition, the return value is not checked.

In `Redemption.sol`, several event emissions are not checked:

- The `RedemptionInitiated` event and the burning of token shares in the `onTransferReceived` function: The burn is checked only via the `Transfer` event and not via the account balance or total supply of tokens.
- The `RedemptionExecuted` event in the `executeRedemption` function
- The `RedemptionCanceled` event and the transfer of token shares in the `cancelRedemption` function: The transfer is checked only via the `Transfer` event and not via the account balance.
- The `EnableOutput` event in the `registerOutput` function

Recommendations

Short term, fix the test suite and ensure that the events are correctly tested.

Long term, modify the test suite to consider adversarial cases and help detect malfunctioning tests early. Use linters or other automated analysis tools to detect incorrect statements in the tests.

2. `publishPrice` can modify an existing price or lock the oracle

Severity: **Informational**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-SPIKO-2
Target: `contracts/oracle/Oracle.sol`	

Description

The `Checkpoints` library contract's `push` function allows operators to update the latest price pushed to the oracle, once it has been set, by submitting a new transaction with the same key value as the last addition (for this particular oracle, the key is the `timepoint` parameter).

The Spiko team's repository issue tracker mentions a similar concern, but the case of overwriting the latest value is not considered.

Additionally, as warned in the `Checkpoints` library contract, if the maximum value is accidentally set for the key, the checkpoint will be effectively disabled.

```
 57    function publishPrice(uint48 timepoint, uint208 price) public restricted()
returns (uint80) {
 58        uint80 roundId = _history.length().toUint80();
 59        _history.push(timepoint, price);
 60
 61        emit Update(timepoint, price.toInt256(), roundId);
 62
 63        return roundId;
 64    }
```

Figure 2.1: The `publishPrice` function (`contracts/oracle/Oracle.sol`)

Exploit Scenario

An account with oracle operator permissions sends a new price value to the oracle. Later, another oracle operator updates the oracle again but overwrites the latest price value by providing the same `timepoint` parameter.

Since the Spiko oracle updates are meant to be predictable (i.e., always happening at the same time on working days), a third-party protocol is not aware of the second update and temporarily operates on an incorrect oracle price.

Alternatively, the oracle operator accidentally sets `timepoint` to `2**48 - 1`, which disables the oracle because it is unable to receive additional updates.

Recommendations

Short term, document whether this behavior is expected and, if it is not, prevent oracle updates that have the same timestamp as the last update. To avoid locking the oracle, limit the value of `timepoint` to the current timestamp plus or minus a security margin.

Long term, to avoid edge cases, document the expected value boundaries for critical functions.

3. Tokens can be locked in the Redemption contract

Severity: **Low**	Difficulty: **Low**
Type: Data Validation	Finding ID: TOB-SPIKO-3
Target: `contracts/token/Redemption.sol`	

Description
The `Redemption` contract uses the ERC-1363–compliant `onTransferReceived` callback function for incoming token transfers that use the `transferAndCall` or `transferFromAndCall` functions.

However, it is possible for a user to manually send tokens using ERC-20's `transfer` function (or for a third-party contract, using the `transferFrom` function), which will lock the tokens in the `Redemption` contract. We are aware that an operator can burn these tokens and mint them back again to the allowlisted user's account, but we are not aware of the consequences of these actions with respect to the EU regulations.

Exploit Scenario
Alice, a Spiko token holder, wants to redeem her shares. Not knowing of the existence of ERC-1363, she sends the tokens to the `Redemption` contract using ERC-20's `transfer` function. Her tokens become locked and cannot be recovered.

An administrator must burn the tokens and mint them back into her account.

Recommendations
Short term, give operators a way to recover tokens stuck in the `Redemption` contract without having to burn and mint them, or provide administrators or operators with an alternative solution, such as an infinite approval from the `Redemption` contract.

Long term, document this behavior and make sure users are aware of the risk.

4. No minimum amount required for redemption	
Severity: **Low**	Difficulty: **Low**
Type: Data Validation	Finding ID: TOB-SPIKO-4
Target: `contracts/token/Redemption.sol`	

Description
There is no limit on the minimum amount of tokens that must be transferred for redemption, so users can generate events with zero-value or dust-value transfers.

The tokens provided in the test suite that correspond to USD- and EUR-nominated shares (spUSD and spEUR) have nine decimals each. Since the user can decide to redeem shares for fiat currency or for stablecoins with fewer decimals (such as USDC), it is possible for token dust to be unredeemable, becoming lost in the process and affecting the total supply of tokens—and therefore the NAV of the shares.

Exploit Scenario
Alice, an allowlisted user who has some shares in her account, decides to transfer dust amounts of her shares to be redeemed for fiat currency, which usually has two decimals. The Spiko team receives numerous `RedemptionInitiated` events for small values.

In an extreme case, she decides to generate a high number of events, via low-amount (or zero-amount) transfers, causing the team an unexpected amount of administrative work to deal with the requests.

Recommendations
Short term, evaluate and document whether this behavior is expected. If it is not, add a minimum token transfer amount for starting the redemption process.

Long term, add the relevant tests for edge cases, where the user triggers unexpected behavior either accidentally or on purpose.

A. Vulnerability Categories

The following tables describe the vulnerability categories, severity levels, and difficulty levels used in this document.

Vulnerability Categories	
Category	**Description**
Access Controls	Insufficient authorization or assessment of rights
Auditing and Logging	Insufficient auditing of actions or logging of problems
Authentication	Improper identification of users
Configuration	Misconfigured servers, devices, or software components
Cryptography	A breach of system confidentiality or integrity
Data Exposure	Exposure of sensitive information
Data Validation	Improper reliance on the structure or values of data
Denial of Service	A system failure with an availability impact
Error Reporting	Insecure or insufficient reporting of error conditions
Patching	Use of an outdated software package or library
Session Management	Improper identification of authenticated users
Testing	Insufficient test methodology or test coverage
Timing	Race conditions or other order-of-operations flaws
Undefined Behavior	Undefined behavior triggered within the system

Severity Levels	
Severity	**Description**
Informational	The issue does not pose an immediate risk but is relevant to security best practices.
Undetermined	The extent of the risk was not determined during this engagement.
Low	The risk is small or is not one the client has indicated is important.
Medium	User information is at risk; exploitation could pose reputational, legal, or moderate financial risks.
High	The flaw could affect numerous users and have serious reputational, legal, or financial implications.

Difficulty Levels	
Difficulty	**Description**
Undetermined	The difficulty of exploitation was not determined during this engagement.
Low	The flaw is well known; public tools for its exploitation exist or can be scripted.
Medium	An attacker must write an exploit or will need in-depth knowledge of the system.
High	An attacker must have privileged access to the system, may need to know complex technical details, or must discover other weaknesses to exploit this issue.

B. Code Maturity Categories

The following tables describe the code maturity categories and rating criteria used in this document.

Code Maturity Categories	
Category	**Description**
Arithmetic	The proper use of mathematical operations and semantics
Auditing	The use of event auditing and logging to support monitoring
Authentication / Access Controls	The use of robust access controls to handle identification and authorization and to ensure safe interactions with the system
Complexity Management	The presence of clear structures designed to manage system complexity, including the separation of system logic into clearly defined functions
Cryptography and Key Management	The safe use of cryptographic primitives and functions, along with the presence of robust mechanisms for key generation and distribution
Decentralization	The presence of a decentralized governance structure for mitigating insider threats and managing risks posed by contract upgrades
Documentation	The presence of comprehensive and readable codebase documentation
Low-Level Manipulation	The justified use of inline assembly and low-level calls
Testing and Verification	The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage
Transaction Ordering	The system's resistance to transaction-ordering attacks

Rating Criteria	
Rating	**Description**
Strong	No issues were found, and the system exceeds industry standards.
Satisfactory	Minor issues were found, but the system is compliant with best practices.
Moderate	Some issues that may affect system safety were found.

Weak	Many issues that affect system safety were found.
Missing	A required component is missing, significantly affecting system safety.
Not Applicable	The category is not applicable to this review.
Not Considered	The category was not considered in this review.
Further Investigation Required	Further investigation is required to reach a meaningful conclusion.

C. Mutation Testing

This appendix outlines how we conducted mutation testing and highlights some of the most actionable results.

At a high level, mutation tests make several changes to each line of a target file and rerun the test suite for each change. Changes that result in test failures indicate adequate test coverage, while changes that do not cause tests to fail indicate gaps in test coverage. Although mutation testing is a slow process, it allows auditors to focus their review on areas of the codebase that are most likely to contain latent bugs, and it allows developers to identify and add missing tests.

We used universalmutator to conduct our mutation testing campaign. Although this tool is language-agnostic, it provided valuable insights into the Spiko smart contracts' Hardhat-based tests. This tool can be installed with the following command:

```
pip install universalmutator
```

Figure B.1: The command used to install universalmutator

Once installed, a mutation campaign can be run against all Solidity source and test files in the Spiko repository, except for the mock contracts, using a Bash script. Figure B.2 shows a minimal example of a mutation script:

```
find contracts \
  -name '*.sol' \
  -not -path '*/mocks/*' \
  -print0 | while IFS= read -r -d '' file
do
  log="mutants/$(basename "$file").log"
  mutate "$file" --cmd "timeout 30s npx hardhat test" > ".tmp.log"
  head -n 4 ".tmp.log" > "$log"
  tac ".tmp.log" | sed -n '/.*break;\.\.\.VALID/{N;p;} ;
/.*continue;\.\.\.VALID/{N;p;} ; /.*\.\.\.VALID/p; ' | tac >> "$log"
  tail -n 4 ".tmp.log" >> "$log"
done
```

Figure B.2: An example Bash script that can be used to run a full mutation testing campaign against all source files in the target repository

Consider the following notes about the above Bash script:

- The overall runtime of the script against the repository is approximately two hours on a consumer-grade laptop. However, running a mutation testing campaign that targets a single contract or test suite will take less time.

- The `--cmd` argument given to the `mutate` executable (provided by universalmutator) designates the test command. Given the long runtime, a timeout is enforced to prevent any single test from stalling the campaign.
- The `tac` and `sed` commands remove all invalid mutants from the output of `mutate` for quicker and easier analysis of the results. The `head` and `tail` tools are used to preserve the analysis summaries.
- We used a custom set of Solidity mutation rules that provide more results than the default set and include relatively more false positives. For more common use of mutation tests, the default Solidity ruleset is sufficient and provides a more user-friendly balance of test coverage assessment and false-positive rate.
- To ensure that the target contract is appropriately recompiled after each mutation, we recommend replacing the test script of the `package.json` file within the target repository with the line shown in figure B.3.

```
"test": "rm -rf ./artifacts ./cache && hardhat compile --force && hardhat test",
```

Figure B.3: Replace the test script of the `package.json` file with this line to ensure that the target contract is recompiled after each mutation.

An illustrative subset of results for the `Redemption` contract is shown in figure B.4.

```
*** UNIVERSALMUTATOR ***
MUTATING WITH RULES: custom-solidity.rules
SKIPPED 59 MUTANTS ONLY CHANGING STRING LITERALS
PROCESSING MUTANT: 19:     IERC1363Receiver,  ==>      /*IERC1363Receiver,*/...VALID [written
to mutants/Redemption/Redemption.mutant.0.sol]
PROCESSING MUTANT: 52:         _disableInitializers();  ==>
/*_disableInitializers();*/...VALID [written to mutants/Redemption/Redemption.mutant.11.sol]
PROCESSING MUTANT: 103:         emit RedemptionInitiated(id, user, input, output, value, salt);
==>          /*emit RedemptionInitiated(id, user, input, output, value, salt);*/...VALID
[written to mutants/Redemption/Redemption.mutant.17.sol]
PROCESSING MUTANT: 130:         input.burn(address(this), inputValue);  ==>
/*input.burn(address(this), inputValue);*/...VALID [written to
mutants/Redemption/Redemption.mutant.21.sol]
PROCESSING MUTANT: 133:         emit RedemptionExecuted(id, data);  ==>          /*emit
RedemptionExecuted(id, data);*/...VALID [written to
mutants/Redemption/Redemption.mutant.22.sol]
PROCESSING MUTANT: 158:         input.safeTransfer(user, inputValue);  ==>
/*input.safeTransfer(user, inputValue);*/...VALID [written to
mutants/Redemption/Redemption.mutant.26.sol]
PROCESSING MUTANT: 161:         emit RedemptionCanceled(id);  ==>          /*emit
RedemptionCanceled(id);*/...VALID [written to mutants/Redemption/Redemption.mutant.27.sol]
PROCESSING MUTANT: 177:         emit EnableOutput(input, output, enable);  ==>          /*emit
EnableOutput(input, output, enable);*/...VALID [written to
mutants/Redemption/Redemption.mutant.28.sol]
512 INVALID MUTANTS
0 REDUNDANT MUTANTS
```

Figure B.4: Abbreviated output from the mutation testing campaign on `Redemption.sol`

False positives and duplicates have been removed, and the output has been reformatted slightly (e.g., differences highlighted) for ease of review. To illustrate the types of gaps in test coverage that a mutation testing campaign can help identify, some results are provided as illustrative examples, but more results are present in the full results file. The following changes to the `Redemption` contract can be made without triggering any test failures:

- Line 19: Removing (or commenting out) the `IERC1363Receiver` interface from the inheritance list for the `Redemption` contract has no effect on the test suite or in the contract's compilation.
- Line 52: Removing (or commenting out) the `_disableInitializers` function from the `Redemption` constructor has no effect on the test suite or in the contract's compilation. This is likely because no tests interact directly with the implementation.
- Lines 103, 130, 133, 158, 161, and 177: Commenting out all event emissions mentioned in these lines has no effect on the tests. This was a surprising result because the test suite explicitly expects events to be emitted. Further investigation led to one of the findings in this report.

We recommend that the Spiko team thoroughly review and evaluate the mutation test results and fix the test suite for all valid mutants. Then use a script similar to that provided in figure B.2 to rerun a mutation testing campaign to ensure that the added or modified tests provide adequate coverage.

The full set of mutation test results can be provided to Spiko at the end of this engagement.

D. Upgradability Checks with Slither

Trail of Bits developed the `slither-check-upgradability` tool to aid in the development of secure proxies; it performs safety checks relevant to both upgradeable and immutable `delegatecall` proxies. Consider using this tool during the development of the Spiko smart contracts' codebase.

- Use `slither-check-upgradeability` to check for issues such as a corrupted storage layout between the previous and new implementations.

```
slither-check-upgradeability . ContractV1 --new-contract-name ContractV2
```

Figure C.1: An example of how to use `slither-check-upgradeability`

For example, if a variable `a` is incorrectly added in the new implementation before other storage variables, `slither-check-upgradeability` will issue a warning like the one shown in figure C.2.

```
...
INFO:Slither:
Different variables between ContractV1 (contracts/versions/ContractV1.sol#9-54)
and ContractV2 (contracts/versions/ContractV2.sol#11-133)
        ContractV1.__gap (contracts/versions/ContractV1.sol#15)
        ContractV2.a (contracts/versions/ContractV2.sol#20)
Reference:
https://github.com/crytic/slither/wiki/Upgradeability-Checks#incorrect-variables-w
ith-the-v2
...
```

Figure C.2: Example `slither-check-upgradeability` output

- Use `slither-read-storage` with the new implementation to manually check that the expected storage layout matches the previous implementation. Running the command shown in figure C.3 for the previous and new implementations will list the storage locations for variables in both versions. Make sure that variables in the previous implementation have the same slot number in the upgraded version.

```
slither-read-storage . --contract ContractV1 --table
```

Figure C.3: An example of how to use `slither-read-storage`

- If gaps are used in parent contracts, check the storage layout with `slither-read-storage` after adding new functionality and decrease the gap size accordingly. Make sure that storage is not overwritten or shifted in child contracts.

- Simulate the upgrade with a local mainnet fork, verify that all storage variables have the expected values, and run the tests on the new implementation.

E. Incident Response Recommendations

This section provides recommendations on formulating an incident response plan. More information can be found on our Incident response guidelines page.

- **Identify the parties (either specific people or roles) responsible for implementing the mitigations when an issue occurs (e.g., deploying smart contracts, pausing contracts, upgrading the front end, etc.).**
- **Document internal processes for addressing situations in which a deployed remedy does not work or introduces a new bug.**
    - Consider documenting a plan of action for handling failed remediations.
- **Clearly describe the intended contract deployment process.**
- **Outline the circumstances under which Spiko will compensate users affected by an issue (if any).**
    - Issues that warrant compensation could include an individual or aggregate loss or a loss resulting from user error, a contract flaw, or a third-party contract flaw.
- **Document how the team plans to stay up to date on new issues that could affect the system; awareness of such issues will inform future development work and help the team secure the deployment toolchain and the external on-chain and off-chain services that the system relies on.**
    - Identify sources of vulnerability news for each language and component used in the system, and subscribe to updates from each source. Consider creating a private Discord channel in which a bot will post the latest vulnerability news; this will provide the team with a way to track all updates in one place. Lastly, consider assigning certain team members to track news about vulnerabilities in specific system components.
- **Determine when the team will seek assistance from external parties (e.g., auditors, affected users, other protocol developers) and how it will onboard them.**
    - Effective remediation of certain issues may require collaboration with external parties.
- **Define contract behavior that would be considered abnormal by off-chain monitoring solutions.**

It is best practice to perform periodic dry runs of scenarios outlined in the incident response plan to find omissions and opportunities for improvement and to develop "muscle memory." Additionally, document the frequency with which the team should perform dry runs of various scenarios, and perform dry runs of more likely scenarios more regularly. Create a template to be filled out with descriptions of any necessary improvements after each dry run.

F. Fix Review Results

When undertaking a fix review, Trail of Bits reviews the fixes implemented for issues identified in the original report. This work involves a review of specific areas of the source code and system configuration, not comprehensive analysis of the system.

On November 16, 2023, Trail of Bits reviewed the fixes and mitigations implemented by the Spiko team for the issues identified in this report. We reviewed each fix to determine its effectiveness in resolving the associated issue.

The Spiko team provided fixes and feedback in its GitHub repository as comments and pull requests. These were manually reviewed to determine whether the issues mentioned in this report were addressed or resolved.

In summary, of the four issues described in this report, Spiko has resolved two issues and has not resolved the remaining two issues. Spiko has accepted the risk of these other two issues. For additional information, please see the Detailed Fix Review Results below.

ID	Title	Status
1	Event emissions are ignored in the provided test suite	Resolved
2	`publishPrice` can modify an existing price or lock the oracle	Unresolved
3	Tokens can be locked in the Redemption contract	Unresolved
4	No minimum amount required for redemption	Resolved

Detailed Fix Review Results

TOB-SPIKO-1: Event emissions are ignored in the provided test suite
Resolved in PR #11. The pull request fixes the issues in the `await/expect` statements in the test suite and adds coverage for ERC-1363 calls and `Ownable` contracts' ownership changes.

For functions in the `Masks` library that were added for completeness and are not used by the project, comments were added to indicate this situation and to clarify that there are no tests implemented for them.

TOB-SPIKO-2: publishPrice can modify an existing price or lock the oracle
Unresolved. The issue has been acknowledged by the Spiko team, who provided the following context for this finding's fix status:

> *We are aware of the behavior flagged in the issue, which is expected at this stage.*
>
> *We indeed decided to keep the business logic in this oracle contract as limited as possible in order to be able to accommodate different types of tokenized securities in the future (with different volatility, price update frequency, etc.). The oracle update process will be centralized: a third party will evaluate the token price and then it will be published both on- and off-chain (including to competent authorities).*
>
> *Depending on the usage of this oracle by other protocols, we might implement an intermediate smart contract that will enforce specific business logic in the `publishPrice` function (say, for instance: enforce update window to be larger than 23 hours and lower than 1 week).*

TOB-SPIKO-3: Tokens can be locked in the Redemption contract
Unresolved. The issue has been acknowledged by the Spiko team, who provided the following context for this finding's fix status:

> *This is something that was consciously left out. The original idea was that users can always "lose" ERC-20 tokens by transferring to an invalid address.*
>
> *However, that is not true in the context of Spiko because the whitelist is supposed to prevent (or limit the risk of) loss of funds through transfers to invalid addresses.*
>
> *For a `recover` function to work, it should compare `token.balanceOf(address(this))` with the expected number (sum of the currently pending redemptions). This is not hard to do but adds expensive SSTOREs.*

Additionally, the following comment was added to the issue to explain the reasoning behind their decision:

We considered the following two options:

- *Implement a way for operators to recover tokens "stuck" in the `Redemption` contract. This would solve the issue but would increase the gas costs for all transactions in the contract because of the necessity to keep track of the tokens received with the wrong function. We have estimated the gas costs increase between +5,2% and +19,2% for the function `onTransferReceived` and +0,5% and +6,4% for the function `executeRedemption`, depending on cases.*

- *Leave the smart contract as is, which means, as detailed in the issue, that a standard ERC-20 transfer (without `transferAndCall` or `transferFromAndCall`) would lock the transferred tokens in the `Redemption` contract.*

Given the fact that we'll provide users with a UI that will implement the right redemption procedure, we estimate the likeliness and the frequency of the "wrong" redemptions to be quite low. We therefore believe it to be more efficient in terms of gas to leave the smart contract as is and perform an admin `burn`/`mint` when users get the redemption wrong. If we end up having underestimated the frequency of these "wrong" redemptions, we will update the `Redemption` contract to implement the first option above.

TOB-SPIKO-4: No minimum amount required for redemption
Resolved in commit 1bc2965. The team added a new mapping to store the minimum redemption amounts for each token. Additionally, a privileged function for setting the minimum amounts was added, and a new event is emitted when the values are changed. Finally, the test suite was updated with cases that cover the new functionality.

G. Fix Review Status Categories

The following table describes the statuses used to indicate whether an issue has been sufficiently addressed.

Fix Status	
Status	**Description**
Undetermined	The status of the issue was not determined during this engagement.
Unresolved	The issue persists and has not been resolved.
Partially Resolved	The issue persists but has been partially resolved.
Resolved	The issue has been sufficiently resolved.