Whales DMCC Nominators Contract

Security Assessment

March 27, 2025

Prepared for:
Sergei Iudin
Whales DMCC

Prepared by: **Elvis Skoždopolj and Guillermo Larregay**

Table of Contents

Project Summary

Contact Information

The following project manager was associated with this project:

Jeff Braswell, Project Manager
jeff.braswell@trailofbits.com

The following engineering director was associated with this project:

Jim Miller, Engineering Director, Blockchain & Cryptography
james.miller@trailofbits.com

The following consultants were associated with this project:

Elvis Skoždopolj, Consultant
elvis.skozdopolj@trailofbits.com

Guillermo Larregay, Consultant
guillermo.larregay@trailofbits.com

Project Timeline

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

Date	Event
February 26, 2025	Pre-project kickoff call
March 10, 2025	Status update meeting #1
March 14, 2025	Delivery of report draft
March 14, 2025	Report readout meeting
March 27, 2025	Delivery of final comprehensive report
May 12, 2025	Completion of fix review

Executive Summary

Engagement Overview

Whales DMCC engaged Trail of Bits to review the security of the `nominators` smart contract and related contracts. The `nominators` contracts manage a pool of nominators' stakes for validator participation, allowing multiple users to combine their TON coins into a single validator stake.

A team of two consultants conducted the review from March 3 to March 14, 2025, for a total of four engineer-weeks of effort. The component highlighted in this report was reviewed in the first week of the engagement, for a total of two engineer-weeks of effort. Our testing efforts focused on review of the privileged roles, their privileges, and their limitations; analysis of potential race conditions and gas consumption; and analysis of bounceable messages, bouncing conditions, and reversal of state.

With full access to source code, we performed static testing of the target, using manual processes. Because the `nominators` codebase is a complex system, several components such as the back end and front end were out of scope for the audit, and their interactions with the contracts were not specified. The off-chain parts of the system along with key management practices were not reviewed. Due to time constraints, we did not fully review the `nominators` contracts. The full list of coverage limitations can be found in the Project Coverage section.

Observations and Impact

The `nominators` codebase is well structured into smaller components; however, the nonstandard compiling procedure makes it difficult to review and will make it more difficult to maintain. We did not have access to any documentation apart from some inline code documentation; such documentation would have improved the reviewability of the code.

The access controls are robust, the codebase contains permissioned operations, and it is not designed to be fully decentralized. We discovered two issues related to privileged role misuse (TOB-WHC-1 and TOB-WHC-4) and one instance where using a two-step process for role updates would provide higher security guarantees (TOB-WHC-2). Limiting the powers of privileged roles by implementing timelocks or user opt-out mechanisms, along with creating user-facing documentation outlining the risks related to interacting with the system, could be beneficial. Writing a system specification detailing behavior and constraints would go a long way toward improving the codebase.

The codebase contains validation in order to prevent issues related to race conditions; however, since a lot of the operations rely on the same state, we did not manage to fully review all race condition scenarios during this security assessment.

Recommendations

Based on the codebase maturity evaluation and findings identified during the security review, Trail of Bits recommends that Whales DMCC take the following steps:

- **Remediate the findings disclosed in this report.** These findings should be addressed as part of a direct remediation or any refactor that may occur when addressing other recommendations.
- **Create user- and developer-facing documentation.** The documentation that we had access to is sparse, consisting of mostly inline code comments. Proper code documentation that includes message flows, user stories, and sequence diagrams should be created in order to make further development, review, and maintenance easier.
- **Refactor the `nominators` contract.** The `nominators` contract does not use `include` compiler directives; instead, it relies on a script to create a single contract from the necessary dependencies. Using `include` statements will make the contract easier to review and maintain.
- **Retroactively create a system specification.** Creating a system specification that includes expected system behavior and operation constraints would help with discovering and preventing the reintroduction of security issues and would help in enforcing important operation constraints.
- **Perform another security review after the findings are addressed and the code is refactored.** Complex contracts such as `nominators` should be reviewed in depth and with better access to documentation and back-end processes.

Finding Severities and Categories

The following tables provide the number of findings by severity and category.

EXPOSURE ANALYSIS

Severity	*Count*
High	0
Medium	1
Low	0
Informational	5
Undetermined	0

CATEGORY BREAKDOWN

Category	*Count*
Data Validation	5
Denial of Service	1

Project Goals

The engagement was scoped to provide a security assessment of the Whales DMCC `nominators` smart contracts. Specifically, we sought to answer the following non-exhaustive list of questions:

- Can user funds become permanently locked?
- Can user funds be stolen?
- Can the controller be prevented from sending stake to the elector?
- Can stake recovery be prevented?
- Can stake recovery be executed out of order?
- Can the profit or loss be incorrectly distributed among the nominators?
- Can the contract owner impact nominator funds or profit by changing the financial properties of the system?
- Can a nominator gain a profit without having their balance staked?

Project Targets

The engagement involved reviewing and testing the target listed below.

Nominators Pool Contract

Repository	https://github.com/tonwhales/nominators-v2
Version	`6f448cd0f34f873b0b03cf5ff60c36d080ed7d58`
Type	FunC
Platform	TON

Project Coverage

This section provides an overview of the analysis coverage of the review, as determined by our high-level engagement goals. Our approaches included the following:

- Manual review of the main message flows in the `nominators` contract, including the following:
    - A review of race conditions considering state updates in multi-message flows, including race conditions related to elector responses
    - A review of the lock and unlock mechanism, looking for ways to bypass it or use it for a denial-of-service attack
    - A review of the profit distribution mechanism to see if users can unfairly gain a larger amount of profit, if a user can avoid a penalty, or if a user whose stake was never deposited can gain a profit
    - A review of the roles, their privileges, and their limitations, including a manual analysis of the ability to use members' funds and of the overall funds handling code for any potential issues
    - Analysis of bounceable messages, bouncing conditions, and reversal of state
    - Review of message values and gas consumption
- Manual review of the testing suite to gain context on expected system behavior

Coverage Limitations

Because of the time-boxed nature of testing work, it is common to encounter coverage limitations. The following list outlines the coverage limitations of the engagement and indicates system elements that may warrant further review:

- We did not have access to specific documentation about the system. The team provided a high-level description of the system, how the contracts interact with each other, and the actual repositories.
- The front end, user interface, back end, and all other components of the system not mentioned in the coverage section were not in scope.
- We did not consider front-end checks or validations.
- All additional files in the project, such as deployment scripts, tests, wrappers, and any non-FunC files, were not part of the scope and, therefore, were not reviewed.
- The following are coverage limitations in the `nominators-v2` repository:

- The elector and vanity contracts were considered out of scope and were only briefly reviewed to gain context on the `stake_send` and `stake_recover` operations.
- The `withdraw_unowned` and `force_kick` operations were not reviewed in depth.
- While race conditions between the `stake_send` and `stake_recover` operations were reviewed, the contract could use further review of all possible race conditions.

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	No arithmetic issues were discovered during the review. The contracts use simple formulas for balance tracking; however, it would be beneficial to document each formula in the code.	**Satisfactory**
Auditing	We are not aware of any off-chain monitoring system that can detect or prevent malicious behavior. The team has not provided documentation about any incident response plans. Functions do not emit external messages or logs when critical storage state is modified.	**Further Investigation Required**
Authentication / Access Controls	The codebase has clearly defined roles, and access controls are correctly implemented for all privileged functions. Some roles can be changed or revoked; however, there are cases where no two-step procedures are implemented for role transfers (TOB-WHC-2). We are not aware of how privileged accounts are protected, especially the ones controlled by the back end. For each non-user account with privileges, the documentation should specify who has access to it, if it is a single key or a multisignature wallet, how it is protected, and what the procedure is if the key is leaked.	**Moderate**
Complexity Management	The way the `nominators-amalgam` contract is created is nonstandard and should be rewritten to use `include` statements that are easier to debug, understand, and parse by both users and IDEs.	**Moderate**

Cryptography and Key Management	We did not review any code related to this category.	**Not Applicable**
Decentralization	None of the contracts in scope are meant to be decentralized. There is one issue in the `nominators` contract (TOB-WHC-1) related to privileged actors getting access to user funds. Additionally, the controller has power over which depositors can have their stake approved (TOB-WHC-4). It would be beneficial to clearly document the privileged role powers and how they can impact user funds; this should be made available as part of the user-facing documentation.	**Weak**
Documentation	We had no access to documentation during the review. The code comments are insufficient and can be improved. The documentation should include architecture descriptions and diagrams, message flow diagrams, and user stories. All critical components should be identified and their risks clearly described. Retroactively writing a system specification would make it easier to discover security issues and guide the improvement of the testing suites.	**Weak**
Low-Level Manipulation	The contracts in scope do not use any low-level manipulation.	**Not Applicable**
Testing and Verification	The test suites are limited in general and consider only "happy paths." Some tests do not run out of the box. Due to time constraints, the testing suites were reviewed only to gain context on message flows and expected system behaviors.	**Further Investigation Required**
Transaction Ordering	The codebase contains multiple operations that share state and could potentially lead to race conditions; we did not manage to fully review all race condition scenarios and protections during this security assessment.	**Further Investigation Required**

Summary of Findings

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

ID	Title	Type	Severity
1	Lack of bounds for fees	Data Validation	**Medium**
2	Lack of two-step process for ownership transfer	Data Validation	**Informational**
3	No limit on the number of members	Data Validation	**Informational**
4	Controller can leave users out of staking	Data Validation	**Informational**
5	New deposits clear pending withdrawals	Data Validation	**Informational**

Detailed Findings

1. Lack of bounds for fees

Severity: **Medium**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-WHC-1
Target: `nominators-v2/src/contract/modules/op-common.fc`, `nominators-v2/src/contract/nominators.fc`	

Description

The `op::update` operation allows the contract owner to update the withdrawal, deposit, receipt, and pool fees. However, there is no upper bound on the deposit, withdrawal, or receipt fees amount; therefore, these fees could be set to any value.

Additionally, the pool fee upper bound is 100% of the profit. An excessive fee amount (e.g., resulting from a typo) may not be noticed until it causes disruptions. Since the fee is calculated as a percentage of the profits, members can be tricked into staking and then getting no profit.

```
() op_update(int value, slice in_msg) impure {

    [...]

    ;; At least op fee
    throw_unless(error::invalid_message(), new_deposit_fee >= fees::op());
    throw_unless(error::invalid_message(), new_withdraw_fee >= fees::op());
    ;; Must be in 0...10000
    throw_unless(error::invalid_message(), new_pool_fee <= 100 * 100);
    ;; At least receipt price
    throw_unless(error::invalid_message(), new_receipt_price >= fees::receipt());
```

Figure 1.1: The `op_update` function, callable by the contract owner (`nominators-v2/src/contract/modules/op-common.fc#L155–L161`)

Exploit Scenario

Alice, the contract owner, sets the pool fee to 100% instead of 10% by mistake. This goes unnoticed by the protocol team, and all nominator profits are claimed by the owner, resulting in a loss of profit for the nominators and in reputational damage.

Recommendations

Short term, set an upper bound for the deposit, withdrawal, receipt, and pool fees. The upper bound should be high enough to encompass any reasonable value but low enough to catch mistakenly or maliciously entered values that would result in unsustainably high fees.

Long term, carefully document the caller-specified values that dictate the financial properties of the contracts and ensure that they are properly constrained. Consider using a timelock mechanism to give existing members the chance to opt out if they do not agree with the new fees, or making them non-updateable once the pool is created.

2. Lack of two-step process for ownership transfer

Severity: **Informational**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-WHC-2
Target: `nominators-v2/src/contract/modules/op-owner.fc`, `nominators-v2/src/contract/nominators.fc`	

Description

When called, the `op::change_address` operation immediately sets the owner or controller address to a new value. The use of a single step to make such a critical change is error-prone; if the function is called with erroneous input, the results could be irrevocable, leading to permanent loss of important owner functionality.

```
() op_change_address(int value, slice in_msg) impure {

    ;; Parse parameters
    var id = in_msg~load_uint(8);
    var new_address = in_msg~load_msg_addr();
    in_msg.end_parse();

    ;; Throw if new_address is already owner or controller
    if (equal_slices(ctx_owner, new_address) | equal_slices(ctx_controller,
new_address)) {
        throw(error::invalid_message());
    }

    ;; Update address
    if (id == 0) {
        ctx_owner = new_address;
    } elseif (id == 1) {
        ctx_controller = new_address;
    } else {
        throw(error::invalid_message());
    }
```

Figure 2.1: The `op_change_address` function, callable by the contract owner (`nominators-v2/src/contract/modules/op-owner.fc#L1–L20`)

Recommendations

Short term, implement a two-step process for all irrevocable critical operations. The `treasure-v0` contract from the holders repository uses a two-step process for ownership transfer, which can be implemented in the `nominators` contract to resolve this issue.

Long term, identify and document all possible actions that can be taken by privileged accounts, along with their associated risks. This will facilitate reviews of the codebase and prevent future mistakes.

3. No limit on the number of members

Severity: **Informational**	Difficulty: **Low**
Type: Data Validation	Finding ID: TOB-WHC-3
Target: `nominators-v2/src/contract/nominators.fc`, `nominators-v2/src/contract/modules/op-nominators.fc`	

Description
The deposit operation does not have access controls; therefore, any account can become a member by just depositing. The process of adding a new member adds a new entry to the members dictionary in the contract storage.

In TVM contracts, there is a hard limit on the number of entries that a dictionary can store and an increased cost for update operations performed on entries. Additionally, storage fees increase with the number of bytes in a contract's storage.

Therefore, not enforcing a maximum number of members can allow griefing attacks where a high number of accounts generate minimum deposits and increase the size of the dictionary. Such an attack would increase the gas consumption of operations and, depending on the impact of the attack, would require the operator to send several stake control operations or the owner to manually remove the fake members.

Recommendations
Short term, set a maximum number of members in the `nominators` contract.

Long term, implement tests to determine the gas consumption impact when the number of members increases. Make potential users and owners aware of this by improving the documentation.

4. Controller can leave users out of staking

Severity: **Informational**	Difficulty: **Low**
Type: Data Validation	Finding ID: TOB-WHC-4
Target: `nominators-v2/src/contract/nominators.fc`, `nominators-v2/src/contract/modules/op-controller.fc`	

Description

The controller has the ability to accept or ignore deposits, as the `op_controller_accept_stakes` function is privileged. Members have no way of forcing their deposit into stake if the controller decides not to accept it, making them lose on potential profits. Moreover, they do not know if their deposit will be accepted, so they may not know about the opportunity cost they are paying.

```
() op_controller_accept_stakes(int value, slice in_msg) impure {

    ;; Check if enough value
    throw_unless(error::invalid_message(), value >= params::pending_op());

    ;; Check if not locked
    throw_if(error::invalid_message(), ctx_locked);

    ;; Parse message
    var members = in_msg~load_dict();
    in_msg.end_parse();

    ;; Process operations
    var member = -1;
    do {
        (member, var cs, var f) = members.udict_get_next?(256, member);
        if (f) {
            ;; Accept member's stake
            load_member(member);
            member_accept_stake();
            store_member();
        }
    } until (~ f);

    ;; Persist
    store_base_data();
}
```

Figure 4.1: The `members` dictionary is part of the message sent by the controller. (`nominators-v2/src/contract/modules/op-controller.fc#L273-L299`)

Recommendations

Short term, add a mechanism to notify members when their deposit is accepted.

Long term, consider implementing a manual staking process that members can use if a certain time has passed since their deposits and the controller has not accepted them.

5. New deposits clear pending withdrawals

Severity: **Informational**	Difficulty: **Low**
Type: Data Validation	Finding ID: TOB-WHC-5
Target: `nominators-v2/src/contract/nominators.fc`, `nominators-v2/src/contract/modules/model.fc`	

Description

When a member requests a withdrawal, if the current contract balance is not enough to fulfill the payment, the withdrawal takes place in two steps: the available balance is sent to the member, and the remaining requested amount is scheduled to be withdrawn from the staked funds; then, the member has to withdraw it when it is available. The difference is stored in the pending withdrawal balances variables.

However, if the member has a pending withdrawal and initiates a deposit, the pending amount is cleared. This can prevent members from being able to withdraw their full requested amount in the second step, especially when the deposited amount is less than the former pending withdrawal.

```
() member_reset_pending_withdraw() impure {
    set_member_pending_withdraw(0);
    ctx_member_pending_withdraw_all = false;
}

() member_stake_deposit(int value) impure {
    throw_unless(error::invalid_stake_value(), value > 0);

    ;; Update balances
    member_update_balance();

    ;; Reset pending withdrawal
    member_reset_pending_withdraw();

    ;; Add deposit to pending
    ;; NOTE: We are not adding directly deposit to member's balance
    ;;       and we are always confirming acceptance of deposit to a pool
    ;;       via sending accept message. This could be done on- and off-chain.
    ;;       This could be useful to make private nominator pools or black lists.
    ;;       Anyone always could withdraw their deposits though.
    add_member_pending_deposit(value);
}
```

Figure 5.1: The `member_stake_deposit` function resets pending withdrawals. (`nominators-v2/src/contract/modules/model.fc#L58–L79`)

Recommendations

Short term, specify the expected outcome and ensure users are aware of it. User flows should not have unexpected behavior.

Long term, improve the documentation to include all interactions and their expected inputs and outcomes. Create diagrams showing contracts, messages, and interactions.

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 does not apply 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. Code Quality Recommendations

The following recommendations are not associated with specific vulnerabilities. However, they enhance code readability and may prevent the introduction of vulnerabilities in the future.

- Contracts should be refactored to avoid duplication. For example, `if` statements in `distribute_profit()` differ in only one line.
- Variable naming should be consistent. For example, in `op_withdraw()`, the `value` parameter is the TON coins sent with the message; however, in `member_stake_withdraw()`, called from `op_withdraw()`, it represents the user-provided amount to withdraw.
- Underscores should be used when returned values are not used, to avoid cluttering the scope with unused variables. For example, not all variables are used every time `ctx_extras` is unpacked.
- The error message in `op_withdraw` can be misleading: it can also be thrown when the value is too high, but the message does not include that possibility.

D. 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 May 12, 2025, Trail of Bits reviewed the fixes and mitigations implemented by the Whales DMCC team for the issues identified during the audit. The Whales DMCC team has resolved one issue and has elected not to resolve the remaining four issues pertaining to the `nominators` codebase. For additional information, please see the Detailed Fix Review Results below.

ID	Title	Severity	Status
1	Lack of bounds for fees	Medium	Resolved
2	Lack of two-step process for ownership transfer	Informational	Unresolved
3	No limit on the number of members	Informational	Unresolved
4	Controller can leave users out of staking	Informational	Unresolved
5	New deposits clear pending withdrawals	Informational	Unresolved

Detailed Fix Review Results

TOB-WHC-1: Lack of bounds for fees
Resolved in PR #1. The deposit and withdrawal fee now have an upper limit of 10 TON, the receipt fee has an upper limit of 1 TON, and the pool fee has an upper bound of 75% of the calculated profit.

TOB-WHC-2: Lack of two-step process for ownership transfer
Unresolved. The client provided the following context for this finding's fix status:

> *This issue is considered informational. We do not see any danger here.*

TOB-WHC-3: No limit on the number of members
Unresolved. The client provided the following context for this finding's fix status:

> *This issue is considered informational. We don't want to keep the limit of the number of members, as the network conditions may change. Now the contract is expected to bounce a message if the limit is reached (cell depth limit will exceed during dictionary update)*

TOB-WHC-4: Controller can leave users out of staking
Unresolved. The client provided the following context for this finding's fix status:

> *This is an expected behavior of the contract. This issue is considered informational, and it is a feature we use for the pools with allow lists.*

TOB-WHC-5: New deposits clear pending withdrawals
Unresolved. The client provided the following context for this finding's fix status:

> *This is an expected behavior of the contract. We do not consider this as a vulnerability.*

E. 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.

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 and 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 X 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

Trail of Bits performed all activities associated with this project in accordance with a statement of work and an agreed-upon project plan.

Security assessment projects are time-boxed and often rely on information 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 software controls and security properties. 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. A project's time and resource constraints also limit their use.